Helsinki University of Technology Department of Communications and Networking
Teknillinen korkeakoulu Tietoliikenne- ja tietoverkkotekniikan laitos
Espoo 2009 Report 6/2009

FOUNTAIN CODES: PERFORMANCE ANALYSIS AND OPTIMISATION

Tuomas Tirronen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission for public examination and debate in Auditorium S4 at Helsinki University of Technology / Aalto University (Espoo, Finland) on the 4th of March, 2010, at 15 o'clock.

Helsinki University of Technology
Faculty of Electronics, Communications and Automation
Department of Communications and Networking

Teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikenne- ja tietoverkkotekniikan laitos

Distribution:
Helsinki University of Technology
Department of Communications and Networking
P.O. Box 3000
FI-02015 TKK
Tel. +358-9-470 25300
Fax +358-9-470 22474

ISBN 978-952-248-266-2
ISBN 978-952-248-267-9 (pdf)
ISSN 1797-478X
ISSN 1797-4798 (pdf)

Multiprint Oy
Espoo 2009

ABSTRACT OF DOCTORAL DISSERTATION	HELSINKI UNIVERSITY OF TECHNOLOGY P. O. BOX 1000, FI-02015 TKK http://www.tkk.fi

Author Tuomas Tirronen

Name of the dissertation

Fountain Codes: Performance Analysis and Optimisation

Manuscript submitted 7.10.2009	Manuscript revised 17.12.2009
Date of the defence 4.3.2010	
☐ Monograph	☒ Article dissertation (summary + original articles)
Faculty	Faculty of Electronics, Telecommunications and Automation
Department	Department of Communications and Networking
Field of research	Teletraffic theory
Opponent(s)	Prof. Muriel Médard (Massachusetts Institute of Technology, USA)
Supervisor	Prof. Jorma Virtamo (Helsinki University of Technology)
Instructor	

Abstract

The fountain coding principle provides a framework for efficient and reliable data transmission techniques over erasure channels, such as file transmission over the Internet. This thesis presents topics related to the optimisation and performance analysis for different settings where fountain coding methods are applied. We start by reviewing the fountain coding principle on which our own contributions are based. Strategies for both elastic and streaming traffic are considered. The coding schemes are typically modelled as stochastic processes and we analyse them using well-known tools, such as Markov chains and fixed-point iteration. Some of the schemes realise the principles of an ideal digital fountain, while the other sacrifice some characteristics, such as time-independence and the statistical equivalence of the encoded packets.

The description of our own work is divided into two parts. The first part begins by addressing the optimisation of the degree distribution of LT coding, the first universal fountain coding method, for small file sizes. We present exact analysis for LT codes of very small size with some novel results. A simulation based method is presented for the analysis and optimisation of longer codes, up to hundreds of source blocks. We further present a method in which a random linear fountain code is divided into parts and conduct a performance analysis of the system. We propose and analyse two different strategies to overcome the performance degradation caused by the division. The first part ends with the description and optimisation of a systematic, sequential coding scheme in which the sender makes greedy choices concerning the repair packet structure on the basis of his belief about the state of the receiver. We present repair packet degree sequences which result in a low required overhead.

In the second part we will address the problem of achieving a low residual erasure probability for streaming traffic using packet erasure correction. The methods are based on a sliding window. Four different methods are presented differing in how the repair packets are constructed. These codes further differ in the repair packet sending strategy; one code always sends a repair packet deterministically after a window movement, while the others send the repair packets probabilistically.

We conclude that methods inspired by fountain coding provide efficient, yet simple, coding strategies for implementing data transfer in many settings.

Keywords fountain coding, erasure coding, rateless coding, coding theory, Markov chains	
ISBN (printed) 978-952-248-266-2	ISSN (printed) 1797-478X
ISBN (pdf) 978-952-248-267-9	ISSN (pdf) 1797-4798
Language English	Number of pages 109 + 65 pp.
Publisher Department of Communications and Networking / Helsinki University of Technology	
Print distribution Department of Communications and Networking / Helsinki University of Technology	
☒ The dissertation can be read at http://lib.tkk.fi/Diss/	

VÄITÖSKIRJAN TIIVISTELMÄ	TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi

Tekijä Tuomas Tirronen

Väitöskirjan nimi

Suihkulähdekoodit: Suorituskykyanalyysi, optimointi ja sovellukset

Käsikirjoituksen päivämäärä 7.10.2009	Korjatun käsikirjoituksen päivämäärä 17.12.2009
Väitöstilaisuuden ajankohta 4.3.2010	
☐ Monografia	☒ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)

Tiedekunta	Elektroniikan, tietoliikenteen ja automaation tiedekunta
Laitos	Tietoliikenne- ja tietoverkkotekniikan laitos
Tutkimusala	Teleliikenneteoria
Vastaväittäjä(t)	Prof. Muriel Médard (Massachusetts Institute of Technology, USA)
Työn valvoja	Prof. Jorma Virtamo (Teknillinen korkeakoulu)
Työn ohjaaja	

Tiivistelmä

Suihkulähdekoodaus tarjoaa puitteet tehokkaalle ja luotettavalle tiedonsiirrolle poistokanavilla, kuten esimerkiksi tiedoston siirrolle Internetissä. Tässä työssä esitellään optimointi- ja analyysimenetelmiä erilaisissa tilanteissa, joissa hyödynnetään suihkulähdekoodausta. Työn ensimmäisessä osassa esitetään suihkulähdekoodauksen perusteet, joihin työssä kehitetyt menetelmät perustuvat. Esitetyt menetelmät kattavat sekä elastisen että virtaavan liikenteen tapaukset. Koodausmenetelmät mallinnetaan stokastisina prosesseina ja analyyseissa käytetään tunnettuja menetelmiä, kuten Markovin ketjuja ja kiintopisteiteraatioa. Ideaalisen digitaalisen suihkulähteen ominaisuudet saavutetaan joillakin menetelmillä, kun taas joissain tapauksissa on luovuttu esimerkiksi aikariippumattomuudesta ja pakettien tilastollisesta samanarvoisuudesta.

Oman työn osuus aloitetaan esittelemällä menetelmä LT-koodien, ensimmäisten yleisten suihkulähdekoodien, astelukujakauman optimointiin. Työssä esitellään pienille lohkomäärille soveltuva eksakti analyysi. Suuremmille tiedon määrille esitellään simulointeihin perustuva optimointimenetelmä, jonka avulla analyysi onnistuu sadoille lohkoille. Tämän jälkeen esitellään ns. satunnaisen lineaarisen suihkulähteen jako useaan osaan. Tämän jaon johdosta syntyvän epätehokkuuden korjaamiseen esitellään ja analysoidaan kaksi eri menetelmää. Työn tämän osan viimeinen aihe on uuden systemaattisen, sekventiaalisen koodin esittely, jossa korjauspaketit muodostetaan tavalla, jossa lähettäjä ylläpitää omaa uskomustaan vastaanottajan tilasta ja tekee ahneita päätöksiä korjauspakettien muodostamisessa. Korjauspakettien askelukusekvenssit optimoidaan siten, että tarvittavien pakettien lukumäärän odotusarvo minimoituu.

Työn viimeinen osa käsittelee pakettitason koodausmenetelmiä jatkuvalle liikennevirralle. Tavoitteena on saada jäännösvirhetodennäköisyys mahdollisimman pieneksi. Esiteltävät menetelmät perustuvat liukuvan ikkunan käyttöön. Työssä käsitellään neljää erilaista menetelmää, jotka eroavat tavassa, jolla korjauspaketit luodaan. Lisäksi menetelmät eroavat korjauspakettien lähetystavassa: yhdessä tapauksessa korjauspaketti lähetetään aina ikkunan siirtymän jälkeen, kun taas muissa paketti lähetetään tietyllä todennäköisyydellä.

Johtopäätöksenä todetaan, että suihkulähdeperiaatteeseen perustuvilla koodausmenetelmillä on mahdollista toteuttaa tehokkaita ja yksinkertaisia tiedonsiirtomenetelmiä.

Asiasanat	suihkulähdekoodaus, poiston korjaava koodaus, nopeudeton koodaus, koodausteoria, Markovin ketjut	
ISBN (painettu)	978-952-248-266-2	ISSN (painettu) 1797-478X
ISBN (pdf)	978-952-248-267-9	ISSN (pdf) 1797-4798
Kieli	Englanti	Sivumäärä 109 + 65 sivua
Julkaisija	Tietoliikenne- ja tietoverkkotekniikan laitos / Teknillinen Korkeakoulu	
Painetun väitöskirjan jakelu	Tietoliikenne- ja tietoverkkotekniikan laitos / Teknillinen korkeakoulu	
☒ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/		

PREFACE

The first time I heard about the topic of this thesis, fountain coding, was when Professor Jorma Virtamo offered me a chance to work on this topic for my Master's thesis in the Networking Laboratory in 2005. I am glad I took the opportunity, as it gave me the opportunity to continue my studies as a post-graduate student, a path that had always intrigued me. Fortunately, I have not regretted the decision I made back then to pursue a Ph.D. degree.

This dissertation is the result of work that had already begun during the writing of my Master's thesis in 2005. Since then, part of the funding for this work has come from the Pannet and ABI projects funded by the Finnish Funding Agency for Technology and Innovation (TEKES). I have had the great opportunity to be a student in the Graduate School in Electronics, Telecommunications and Automation (GETA) since the autumn of 2006, and it is from there that I have received most of the funding. I also gratefully acknowledge the personal scholarships received from the Nokia Foundation, HPY:n tutkimussäätiö and the Finnish Foundation for Technology Promotion (TES).

I would like to thank with my deepest gratitude the supervisor of my thesis, Professor Jorma Virtamo. Without his enthusiasm for research work and numerous ideas this thesis would not have been possible. It has been a great privilege to be supervised by and to work together with such a talented researcher. I would further like to thank Dr. Esa Hyytiä for many discussions during the initial phase of my work and for co-authoring the first two publications which are part of this thesis. Many thanks also go to Docent Samuli Aalto and William Martin for the help and improvement ideas provided during the writing of the manuscript of this thesis.

The Department of Communications and Networking, previously the Networking Laboratory, has been a great place to work during these years. I would like to thank all of the staff for providing a nice atmosphere. In particular I would like to thank the floorball gang for keeping me at least partly fit. During the years at TKK my office and my office mates have changed a couple of times. Big thank-yous go to Eeva Nyberg-Oksanen, Riikka Susitaival and Aleksi Penttinen for many discussions on other important things in life.

Finally I would like to thank my family and friends for an enjoyable journey and my wife Laura for her love, and for listening (or at least pretending to) to my sometimes endless mumbling about mathematics and my research topics.

Helsinki, Finland, 2.9.2009

Tuomas Tirronen

CONTENTS

LIST OF PUBLICATIONS

[1] E. Hyytiä, T. Tirronen, and J. Virtamo. Optimal degree distribution for LT codes with small message length. In *Proceedings of the 26th IEEE International Conference on Computer Communications,IEEE INFOCOM 2007*, pages 2576–2580, Anchorage, Alaska, USA, May 2007.

[2] E. Hyytiä, T. Tirronen, and J. Virtamo. Optimizing the degree distribution of LT codes with an importance sampling approach. In *Proceedings of the 6th International Workshop on Rare Event Simulation, RESIM 2006*, pages 64–73, Bamberg, Germany, October 2006.

[3] T. Tirronen and J. Virtamo. Performance analysis of divided random linear fountain. In *Proceedings of the Global Telecommunications Conference, IEEE GLOBECOM '07*, pages 520–526, Washington, D.C., USA, November 2007.

[4] T. Tirronen and J. Virtamo. Greedy approach for efficient packet erasure coding. In *Proceedings of the 5th International Symposium on Turbo Codes and Related Topics, Turbo coding 2008*, pages 344–349, Lausanne, Switzerland, September 2008.

[5] Tuomas Tirronen and Jorma Virtamo. Finding fountain codes for real-time data by fixed point method. In *Proceedings of the International Symposium on Information Theory and Applications, ISITA 2008*, pages 1244–1249, Auckland, New Zealand, December 2008.

[6] Tuomas Tirronen and Jorma Virtamo. Performance analysis of sliding window based erasure correction for real-time traffic. In *Proceedings of the 5th Euro-NGI Conference of Next Generation Internet Networks, NGI 2009*, pages 1–8, Aveiro, Portugal, July 2009.

[7] Tuomas Tirronen. Sliding window-based erasure correction using biased sampling. In *Proceedings of the 4th International Conference on Systems and Networks Communications, ICSNC 2009*, pages 144–152, Porto, Portugal, September 2009.

ABBREVIATIONS

3GPP	3rd Generation Partnership Project
ARQ	Automatic repeat request
BEC	Binary erasure channel
DCCP	Datagram congestion control protocol
FEC	Forward error correction
GE	Gilbert-Elliott
LDPC	Low-density parity-check (codes)
LT	Luby transform
PCO	Pre-code only
PEC	Packet erasure channel
RFC	Request for comments
R-S	Reed-Solomon
RFL	Random linear fountain
TCP	Transmission control protocol
UDP	User datagram protocol
WWW	World Wide Web
XOR	Exclusive OR

1 INTRODUCTION

Since the dawn of communications using different kinds of signals, researchers and inventors have come up with methods for correcting the errors which occur in information transmission. One of the most basic forms of error correction is to repeat the transmitted message several times, to make sure the recipient receives the whole message correctly. Instead of the whole message being replicated, the recipient can explicitly ask for specific parts of the message to be sent again, because parts are either completely missing or they were received with some errors (or, at least the receiver believes there is something wrong with the message). Even the most basic forms of communications using electromagnetic signals, such as using maritime signal lights between ships, or Morse code over radio or telegraph lines use these kinds of methods to provide reliability against errors caused by bad or clogged transmission channels, the inability of the receiver to receive the original message in the intended form, or its misinterpretation.

The Internet was originally based on best-effort communication. This means that no guarantees are given by the network that all of the data will be delivered, or on the quality of the service the users or applications will receive. Some transmission protocols, such as the Transmission Control Protocol (TCP) [Pos81], have mechanisms which provide the means to ensure that all of the data are delivered to the recipient intact. The mechanism used by TCP relies on all of the received data packets being acknowledged. When the sender does not receive proper acknowledgements for all of the sent data, he deduces that something is missing and sends the relevant packet again. This strategy works well enough in many scenarios, as can be concluded from the success of different services used on the Internet which rely on all of the data being received correctly, such as World Wide Web (WWW) pages, email, file transfers and so on.

Although asking for missing pieces works, it is not the best method for all situations. It is easy to come up with circumstances where explicitly asking for missing packets would cause performance problems for the server working as the distribution platform. For instance, nowadays a common scenario in the Internet is the distribution of large-sized content to multiple receivers. The content might be movies, games or installation files for operating systems, whose distributors are companies and communities selling, or even providing free downloads of their products over the Internet. These kinds of situations, when there are many downloaders with multiple simultaneous requests for again resending missing parts of the files, could potentially cause network congestion in the physical network hardware or overuse of the serving capacity of the distribution server.

An entirely different kind of approach to error correction, compared to resending missing information, is provided by forward error correction (FEC) coding methods. FEC codes work by using extra redundancy to overcome the problem of missing data. The messages sent using FEC coding are traditionally coded in such a way that the original message block size of k symbols is extended to n, effectively resulting in using a fraction k/n of

the available bandwidth for the actual information transmission. Another option is to send packets of the same size as with the other methods but to send extra packets in addition to the absolute minimum number required. FEC codes require some amount of computational effort in order to encode and decode the sent messages, as at least parts of the received data is not the same as the original data and needs to be decoded. In addition, they typically utilise some kind of finite field arithmetic in their operation.

The main benefit of FEC methods, as their name suggests, is the ability of the code to correct errors without relying on a repeat-type operation. This means that lost information does not need to be asked for again, nor do the received data need to be acknowledged. This kind of operation is advisable, for example, in the content distribution example, or in any kind of situation where the use of a backward channel is not possible or encouraged.

The branches of science studying the means to correct errors in communications are coding theory and information theory. Asking for missing pieces belongs to a broad class of automatic repeat request (ARQ) mechanisms. The class studied in this thesis is the already mentioned FEC codes. Repeating the original message several times is called a repetition code, and is one of the simplest methods to implement forward error correction. All of these methods can be employed on different communication layers. However, in this thesis we specifically study the packet level, where modern FEC methods are applied directly to the packet itself. The most fundamental result in information theory, Shannon's channel capacity theorem [Sha48], basically states that there is no need to use the backward channel to achieve the best possible bandwidth utilisation, and thus the use of one on the Internet is not needed for achieving the best possible performance and use of available resources.

In particular, the topic of this thesis is the analysis of different settings where methods inspired by *fountain coding* are used. In effect, fountain codes are *packet erasure correcting* codes. Erasure correction refers to correcting packet erasures, that is, correction and recovery of full packets erased by the network as a result of network congestion or mechanisms dropping packets with detected bit errors. This is in contrast to *error correction*, where the erroneous bits, usually because of physical imperfections in the communication medium itself, are corrected, by ARQ or FEC methods.

End-to-end data transfer in networks is an example of a scenario, in which erasure codes can be used to provide FEC. The end-to-end path can be modeled as an erasure channel, where a particular packet is either transferred correctly or dropped. The modern-day Internet is an important application for this simple channel model, and packet erasure coding could be used, for example, in conjunction with protocols providing only best-effort service, such as UDP (User Datagram Protocol) [Pos80]. One potential problem with these schemes is that to complete the decoding, the amount of overhead (in addition to the absolute minimum) can grow impractically large if the codes are not designed carefully enough.

In addition to elastic traffic, such as file downloads, we also propose and analyse erasure correction methods for streaming traffic. This is an interesting application of fountain coding-related schemes, as many types of

Internet media today are streamed to a potentially large group of simultaneous users (video streams, Internet radio, etc.).

1.1 Fountain and rateless erasure coding

Fountain codes are a novel and interesting method to provide robustness against packet losses. They were developed especially for such scenarios as multicasting or for delivering large amounts of content to multiple recipients simultaneously. Fountain codes are essentially FEC codes, but instead of the traditional way of specifying the overhead used, i.e., the redundancy used for the erasure correction, before transmission, they typically work by sending packets as long as needed for decoding, with the potential to generate a practically infinite number of different packets. This is called the *rateless* property of the codes.

All of the packets include the same information from a statistical viewpoint; no packet is more important than any other for the recipient. The term 'fountain coding' originates from an analogy to a metaphorical digital fountain. The server is the fountain, spraying packets, like water drops, into the air. The recipient (or several recipients) have buckets for collecting the drops. As long as a bucket gets filled, it does not matter which water drops are used or in which order they arrive to fill the bucket [BLMR98]. Only the amount of water falling in decides if the collection is successful or not.

Many kinds of codes can be constructed and used to approximate the ideal digital fountain principle. In practice, it is hard to satisfy all of the ideal properties of such a fountain. The Luby transform (LT) codes [Lub02] and Raptor codes [Sho06] can be regarded as the state-of-the art fountain codes fulfilling the ideal properties for an asymptotic case (i.e., infinite file length). Other schemes such as the classic Reed-Solomon codes [RS60] can be used to approximate the ideal digital fountain. In this thesis, we will study different settings in which fountain codes and methods inspired by the fountain coding principle are employed. Although not all of the methods are strictly fountain codes, the ideas and inspiration behind the mechanisms that are studied come from the fountain principle. The encoding is based on the same algorithm as in LT codes, namely using XOR operation to add up file blocks into linear combinations which are sent as packets or repair packets. The decoding is also performed using a similar iterative algorithm as with the LT and Raptor codes.

1.2 Contributions and the perspective of this thesis

In this thesis we consider different kinds of settings where fountain coding and fountain type codes are used for packet erasure correction. The actual applications are not specified, but could be, for example, file transmission to one or several receivers, over the Internet. Our viewpoint is to consider the end-to-end paths on the application level. Of the different fountain coding methods, we will study LT codes in particular.

Fountain coding methods like LT and Raptor codes have been proved to be asymptotically optimal. This means that as the file length in blocks tends to infinity, the additional overhead required for decoding tends to

zero. In the first part of our contribution small scenarios are studied, in which the file lengths are some hundreds of blocks at most. We feel that it is important to study and develop fountain coding for these kinds of scenarios, as exact analyses can be made and the results provide some insight into the larger cases, as well.

Also streaming applications are considered, where the residual erasure probability is used as the performance metric. As typical streaming applications need to take into account the fact that some data may be irrecoverably lost during the transmission, the applications often use error concealment and other techniques, which allow the data to be useful even if some parts are lost. The objective of the erasure correction methods proposed in this thesis for streaming traffic is to make the residual erasure probability, i.e., the probability of a block in the stream remaining erased after all possible correction attempts, as low as possible. When this probability is minimised, the coding works as efficiently as possible, and the quality of the actual content of the data in the original stream is improved.

In brief, the contributions of this thesis include the following.

- We find optimal degree distributions for LT codes for cases of three and four file blocks by making an exact analysis of the LT coding process as a Markov chain.
- We optimise LT coding degree distributions for up to 30 message blocks using an exact combinatorial method.
- We propose a simulation-based method for LT degree distribution optimisation for up to hundreds of blocks. The objective function estimate is constructed using ideas from importance sampling theory and optimised using a gradient-based algorithm.
- We analyse a random linear fountain code divided into multiple parts for lower computational complexity. We propose methods to improve the performance loss caused by the division.
- We study a systematic erasure correcting code for channels with a known loss probability, based on updating the belief of the sender about the receiver's status and making greedy decisions in the repair packet generation.
- We propose and optimise erasure correcting codes using a sliding window for real-time traffic. Four different methods are presented and analysed. The methods differ in the way in which the repair packets are generated and, in particular, how the stream blocks are sampled when their inclusion in a repair packet is being considered. We propose methods based on half-window step size, independent sampling probability for the blocks in the current window, and degree-based sampling methods using either uniform or biased sampling in the current window.

Next we will describe in more detail the contributions of this thesis. The presentation is divided into two categories: optimisation for short file lengths and erasure correction based on the fountain principle for streaming (real-time) traffic.

1.2.1 Optimisation for small code lengths

We start by considering optimising the degree distribution, the component of fountain codes determining their performance, for small message lengths of the LT codes. We model the coding process as a Markov chain and solve the optimal degree distributions for toy cases of three and four original file blocks. We proceed by developing a combinatorial approach, which can be used to analyse larger scenarios for up to tens of source blocks. Thus, the results that are presented give optimal degree distributions for LT codes for file sizes below 30 file blocks. We further identify some key properties of these distributions: the importance of the low degrees, the requirement for the presence of a high degree component, and that the degree distribution is quite robust to changes in many directions; only the changes in some principal directions of the optimal degree distributions result in notable deterioration of the performance of the coding.

We continue with the optimisation of LT code degree distributions by presenting a novel idea of using an estimator based on simulations and constructed using the principles of importance sampling. This estimator can be evaluated for a different degree distribution than that originally used in the simulations. Using simulations with a certain degree distribution, a gradient of the estimator is constructed. Then a better degree distribution is searched in the direction of this gradient. The degree distribution is iteratively optimised using a numerical algorithm based on these ideas. The method works reasonably well, but has some numerical difficulties because of the finite simulation sample set. Again, we find that the same key properties are present with the best forms of the degree distributions that are tested. The idea of using a simulation-based estimator which can be evaluated using a different degree distribution is novel, and can potentially have applications in other kinds of optimisation problems as well.

After considering LT codes and their optimisation, we present and analyse two different erasure coding methods for elastic traffic. We consider a random linear fountain code divided into multiple parts, in order to lessen the computational complexity of the decoding, and study how the division affects the overhead required for successful decoding. We propose two different strategies to overcome the deterioration of performance caused by the division, namely using a data carousel-based approach to send packets into different parts and the use of macropackets, which actually represent an LT code applied on top of the divided random code. The macropackets preserve the ideal property of fountain codes that all packets are stochastically identical. However, when channel loss probability is realistically low and the losses are independent, the data carousel-inspired method works better.

As the final method for small code lengths and elastic traffic, we present and analyse a systematic erasure correction method for scenarios in which the channel loss probability is known or estimated. The coding process is started by sending original data blocks as-is once, i.e., the method is systematic. Some of the sent systematic packets are dropped, and after the initial round repair packets are generated and sent to fill the gaps of missing blocks. Using the probability distribution of the number of missing packets as a belief about the situation of the receiver, the sender uses a greedy

criterion to select the blocks to be included in the repair packets. The sender updates his belief based on the degree of the repair packets sent, taking also into account the possible repair packet erasures, and further applies the greedy criterion to select the degree for subsequent repair packets. Thus, the sender calculates a deterministic sequence of degrees for the repair packets. With the previously presented methods we are not able to analyse scenarios with large file lengths, because of state space explosion or other computational complexity issues in the analyses. However, this sequential packet degree calculation works for larger file lengths as well.

1.2.2 Erasure codes for streaming media

The second part of the thesis considers different strategies for correcting erasures when streaming data are transferred. We continue in the spirit of the last method presented for elastic traffic and assume that an estimate for the channel loss probability is available. The original fountain codes are not applicable as-is for streaming traffic, as they were developed to work over large file sizes without considering the possible real-time requirements of the applications at all. Our proposed methods use a sliding window, which ensures that the time constraints of real-time applications are met. All of the methods that are considered are systematic, and thus it is possible to use most of the sent packets as-is without any decoding operations. This is particularly helpful for receivers that do not have the computational capabilities or are otherwise not willing or able to perform decoding. In addition, when the channel loss probability is low or even zero, the decoder does not need to do anything to the received packets; they are useful as-is.

The sliding window is used in two different settings. First, we analyse a case where the window is moved in half-window-sized steps. After the window movement the novel blocks introduced into the window are sent as systematic packets. A repair packet is always sent in a deterministic fashion after the systematic packets. The repair packets are constructed by first choosing a degree for each of the half-windows and combining the corresponding number of randomly selected blocks from the respective half-windows using a bitwise XOR operation.

The second strategy considers probabilistic repair packet generation, in which redundant packets are sent after each systematic packet with a certain probability. This scenario is "continuous"; the sliding window moves one block at a time. We present three different codes, which differ in the way in which the repair packets are generated. First, we propose a method where each of the blocks corresponding to different window locations has its own probability of being sampled into a repair packet. This method does not work as well as, for example, the half-window method, in which the repair packets are generated by degree-based sampling. Furthermore, the analysis, and thus also the optimisation, is based on an independence assumption, which raises the question of whether the repair packet generation really is optimal.

The two other methods studied in this thesis are based on sampling repair packet degrees with probabilistic repair packet sending. The repair packets are generated by first sampling the current window by selecting a packet degree, as in LT coding. The two methods differ in the actual sam-

pling strategy; one of the methods uses uniform sampling after the degree is chosen, whereas the other method allows biased sampling of the blocks. The latter method proves to be more efficient, on a par with the state-of-the-art Raptor coding.

The drawback of all of the proposed methods for streaming traffic is that the codes are not rateless. The optimal code constructs and their rate depend on the targeted residual error probability as well as on an estimate for the channel loss probability and the actual loss patterns, i.e., if the losses are independent or correlated (bursty). However, the performance of the methods seems to be quite insensitive to loss probability variation.

The simulation results with the last two methods reveal some interesting characteristics. They suggest that the optimal degree distributions are of a single-degree type; that is, the optimal repair packets are generated by always choosing the same packet degree. In addition, in the biased sampling case, the optimal sampling probabilities form fixed patterns, in which certain window locations are always sampled. These results are interesting because in many other fountain coding scenarios, such as in LT coding, the degree distributions are more complicated and not trivial at all.

1.3 Terminology

The terminology is neither standardised nor completely uniform in different references, which might cause some problems for the reader. *Fountain code* is a term originally coined by the inventors of the Tornado and LT codes. A more descriptive technical term would be rateless erasure correcting code, which declares what the codes actually are: rateless codes, which correct erasures. On the other hand, the term fountain code includes the metaphor of a digital fountain, which can also be used to explain the function of these types of codes.

During the work behind this thesis we have mostly used the term fountain codes to refer to all kinds of codes which operate similarly to LT codes: the encoding is simple XOR operation between blocks of data chosen randomly using a degree distribution, and decoding is based on an iterative algorithm working through a bipartite graph in order to solve the original data using the received packets. This usage is not strictly correct in all of the cases as, for example, some of the codes presented in the thesis are not rateless in the same sense as true fountain codes. Nevertheless, the methods are inspired by the original fountain idea and approximate the ideal principle, and thus the use of the term fountain code is justified.

Further, when discussing the details of the contributed methods, we refer to the source symbols as source blocks or message blocks. This is to emphasise that the original contiguous data, being a file or stream, are divided into equally-sized segments. Similarly, the encoding symbols, produced by the encoding algorithm from source blocks, are termed packets. When we discuss systematic codes, i.e., codes which send the original data as-is at least once, the redundant packets are called repair packets.

1.4 Outline of the thesis

This thesis is based on seven publications studying different settings in which fountain coding or fountain code-like schemes are used and optimised. Before presenting our own work in detail, we start with a review of the ideas and some theory behind the fountain and rateless coding, in Chapter 2.

Chapter 3 presents optimisation techniques for small length erasure correction codes. First we study LT coding and the optimisation of the degree distribution with exact methods, covered in Publication 1. For longer file sizes we consider the simulation-based approach presented in Publication 2. We discuss the results and the form of optimal degree distributions for LT codes as given by these two methods. After this we present a method, from Publication 3, in which a random linear fountain, reviewed in Chapter 2, is divided. We carry out a performance analysis and discuss how the performance can be improved either by resorting to macropackets, an LT code over the divided random code, or by a data carousel-inspired scheme. The chapter ends with a proposal for a novel systematic coding scheme, in which the sender updates his belief about the state of the receiver and generates the repair packets by making greedy decisions. This is covered in Publication 4.

The second part of the thesis is presented in Chapter 4, where we consider fountain coding inspired schemes for streaming traffic. The codes are systematic, and are based on a sliding window. Four different methods are presented, differing in how the actual repair packets are generated. All of the codes proposed in this chapter are systematic. Scenarios in which repair packets are generated either by sampling packet degrees in half-windows or by giving each window position its own sampling probability are studied first. The optimal strategies are found using a fixed-point iteration. This study is covered in Publication 5. In the remaining two schemes, studied in Publications 6 and 7, the repair packet generation is fully degree-based, with either uniform or biased sampling of the original source blocks in the window.

Each of the chapters ends with a summary of the developments and results provided, and the thesis ends with a brief round-up of the developments in Chapter 5.

2 FOUNTAIN CODING

In this chapter we discuss packet erasure correction in general and review the fountain coding principle. We give examples of codes which can be used to implement the digital fountain, with an emphasis on the LT codes, the first universal fountain codes.

2.1 Basics of packet erasure coding

The reliability of packet-based transmission can generally be provided by two different mechanisms. The goal is to get each packet to its destination intact. This can be achieved either by resending erroneous or dropped packets, or by using some mathematical algorithm to decode the missing data from redundant packets or redundant information added to the sent packets. The first paradigm leads to automatic repeat request (ARQ) types of operations and protocols. Typically, ARQ refers to the use of explicit ARQ methods in lower-layer protocols, which are responsible for the error-control of bit streams across an unreliable link. The ubiquitous TCP [Pos81] is an example of a transport layer protocol which relies on resending packets which are lost during transmission or get routed via such a long route that the sender cannot be sure whether they will arrive at their intended destination. However, this mode of operation is usually employed only in unicast protocols, as the performance of ARQ schemes deteriorates when multicasting over many different paths.

Another way to correct for the missing data is by taking advantage of the data already received. In forward error correction (FEC) techniques this is done by adding redundancy in a controlled way, making it possible for the receiver to recover lost data by taking advantage of the redundant information in the received data. The advantage of FEC schemes is that retransmissions are not required for acquiring the data lost in the channel, and thus a backward channel is unnecessary in many situations. The traditional use of FEC is on the lower layers of communication. FEC codes are also used in mass storage devices for error protection. Software FEC and packet-based erasure correction mean the same thing; the latter term is used later on in this thesis to make the distinction between traditional FEC codes and modern packet-based coding methods.

The discussion on the applicability of FEC methods to packet-based transmission began with the advent of broadband networks and the new possibilities they offered in terms of applications. During the emergence of these possibilities the use of FEC instead of, or in addition to different ARQ schemes was considered [McA90],[SM90]. Later, for example in [NB96] and [Hui96], the authors proposed the use of packet-level FEC in multicast trees instead of relying on retransmission-based schemes and further demonstrated that it is a viable option.

When considering applying FEC codes on the packet level we arrive at the problem that the encoding and decoding procedures are typically handled by software instead of hardware, as is the case when considering tradi-

tional ARQ and FEC codes for example on network links. The traditional FEC algorithms are much more demanding computationally than ARQ; FEC often requires complex operations that possibly use some finite field (Galois field) arithmetics. This is the reason why traditional network protocols work using retransmissions to handle lost data. Simply put, software FEC has not been a viable scheme as the computing power and memory space have not been adequate and efficient FEC algorithms have not existed. Retransmission is easier to implement and requires very little extra computational effort, only some bookkeeping to keep trace of the acquired and missing data.

Modern computing technology, however, makes it possible to implement and operate software-based FEC efficiently. Several studies have been carried out to point out that the encoding and decoding algorithms are fast enough to be used in practise, for example in [Riz97a], [Riz97b] and [Sii08]. Most importantly, the coding methods presented in this thesis utilising fast XOR-based encoding and iterative decoding algorithms render the arguments about computational complexity altogether obsolete. Thus, software FEC or erasure coding is a viable alternative to more traditional schemes. Even the classic Reed-Solomon erasure codes, with their comparatively slow encoding and decoding algorithms can be used in many scenarios efficiently enough.

2.1.1 Erasure channel

The channel model we assume in this thesis is the binary erasure channel (BEC) or its trivial extension, the packet erasure channel (PEC). The models are simple, but still useful, for example, when considering packet based transmission over the Internet. In packet-based transmission the lower communication layers typically take care of bit errors. When bit errors render a data packet useless it is discarded. Another reason for lost packets is network congestion, when the intermediate network nodes between the source and the destination drop packets as a result of limited buffer space.

The input alphabet for BEC is $A_i = \{0, 1\}$, and the possible output symbols are $A_o = \{0, ?, 1\}$, where an *erasure* is denoted by the ?-symbol. The probability that an output symbol corresponds to the sent input symbol is $1-p$. The erasures are independent with a bit erasure probability p. The packet erasure channel works similarly to BEC, but the input and output alphabets can be considered to contain larger chunks of bits instead of just one bit. BEC is depicted in Figure 2.1.

Regardless of the method which is used to provide reliability and resilience to packet erasures (or to provide error correction in general), there is a limit to what can be achieved in terms of erasure-correcting capability. Claude Shannon formulated the basics of information and coding theory in 1948 with an explanation of *channel capacity* [Sha48]: All communications channels have a capacity C and there exists error control codes with rates R, such that when $R < C$ the residual error probability is arbitrarily low. A proof for this can be found, for example, in [CT91].

It is easy to see that the upper bound of the capacity of BEC (and PEC) is $1-p$. Peter Elias, who originally introduced BEC in 1955 [Eli55], proved that the capacity of BEC is indeed exactly $1 - p$, where p is the probability

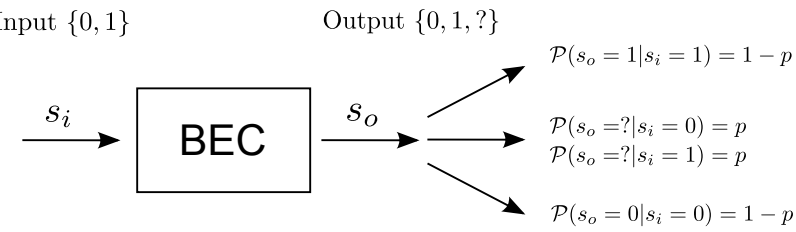


Figure 2.1: Illustration of the binary erasure channel. The packet erasure channel is a trivial extension of this concept.

that a bit sent over the channel is erased. Similarly, for PEC, the capacity is $1 - p$, where p is the packet erasure probability.

For packet-based erasure correction this means that it is possible to send data without any erasures with an effective code rate of $1 - p$. Additionally, Shannon's formulation does not explicitly require any kind of backward channel, and thus it is possible to get arbitrarily close to Shannon's limit with a coding scheme without any retransmission requests. This is an important property and we mainly discuss channels without feedback in this thesis. Ideally, codes approximating the fountain principle, to be introduced in Section 2.2, send enough data through the erasure channel to the recipients that they are able to decode the original message without using any feedback. However, in practice, and especially with only a few recipients, sending acknowledgements might be necessary to stop flooding once the receivers have decoded the data.

In conclusion, we will use the packet erasure channel model without any feedback, such as packet acknowledgements and negative acknowledgements, throughout this thesis. Further, we will not explicitly consider how the coding process is ended, or what the exact mechanism to acknowledge a successful transmission of the whole message is, but instead the analyses are based on the condition that the transmission ends when the decoding procedure is complete.

2.1.2 Traditional erasure coding

We consider a setting where a file is divided into k equally-sized blocks and sent over a data network, which is modelled as a packet erasure channel. A simple example of a single erasure-correcting code is a parity code, where a redundant symbol, the bitwise modulo-2 sum of all of the source symbols, is generated. If one of the symbols is lost during transmission, the missing symbol can be calculated by summing all of the received symbols together. We see that any k of the blocks is sufficient for a complete decoding of the original message. The drawback is that only one erasure can be corrected using this method. Another trivial code is the repetition code, in which a symbol is sent n times over the channel. Here the disadvantage is the very inefficient use of the bandwidth; only a fraction $1/n$ of the sent data are unique. Therefore, better methods operating closer to the Shannon limit have been developed.

A more efficient way to construct an FEC code over erasure channels is to consider a suitable polynomial and send a description of this to the recipients. We can uniquely determine a polynomial $f(x)$ of degree $k-1$ if we know its value at k points. Now, if we use the polynomial to evaluate a set of n distinct values, we get the set $\{(x_i, f(x_i))\}_{i=1}^{n}$, where any selection of k pairs can be used to uniquely determine $f(x)$.

In this way we can define an (n,k) block code, in which k source symbols are used to produce n encoding symbols. Over the erasure channel, we use k file blocks to generate n packets, which are sent over the channel to recipients. For every k information bits, the total amount of transferred data is n bits, resulting in the *code rate*

$$R = \frac{k}{n} \ . \tag{2.1}$$

Thus, the bandwidth required for sending a fixed amount of data is n/k times greater than without using coding at all. An (n,k) block erasure code can recover from a loss of up to $n-k$ encoding symbols. An important feature of all block erasure codes is that any k received encoding symbols (packets) can be used to decode the original message.

Sending packets describing a polynomial is the key idea behind the well-known and widely used Reed-Solomon codes [RS60]. These codes can be used to correct both errors (i.e. bit errors) and erasures. In this thesis, however, we are solely interested in the capabilities of Reed-Solomon coding to correct packet erasures.

Reed-Solomon coding is based on defining polynomials over finite fields (Galois fields). The k blocks composing the original data are encoded into n packets, as discussed above. However, with the finite field arithmetic that is used, the upper bound on n is limited by the size of the field. Typical sizes of the extension fields used in Reed-Solomon coding are 256 and 65536, corresponding to the Galois fields GF(2^8) and GF(2^{16}), respectively. If the source data size is larger than the maximum possible n, the data have to be sent using several segments of n packets.

The maximum sustainable loss probability for the erasure channel is $(n-k)/n$. If the receiver does not receive at least k packets, the Reed-Solomon code cannot recover the original data. Thus, several cycles of packets might be needed. This, in addition to the possible multiple segments of coding, will result in inefficiencies, as the recipient would collect duplicate packets over time, reducing the efficiency of the channel [BLMR98].

An example implementation of Reed-Solomon coding for the erasure channel can be found in [Riz97b]. The coding is implemented using Vandermonde matrices, where the arithmetic operations are implemented as table lookups or simple XOR operations. As the Reed-Solomon codes are *linear*, linear algebra methods can be applied in the coding and decoding processes. The Vandermonde matrix defines the *encoding matrix* $\boldsymbol{G}$ which gives the relation between the encoding symbols $\boldsymbol{y} = (y_0 \quad y_1 \quad \cdots \quad y_{n-1})$ and the source symbols $\boldsymbol{x} = (x_0 \quad x_1 \quad \cdots \quad x_{k-1})$ by the linear mapping

$$\boldsymbol{y} = \boldsymbol{G}\boldsymbol{x} \ . \tag{2.2}$$

The encoding complexity per sent packet is then $O(k)$, or $O(nk)$ for all the packets generated. The decoding can be performed by inverting the encoding matrix $\boldsymbol{G}$, i.e., solving $\boldsymbol{x}$ in the linear system (2.2). In its most basic form, the linear system can be solved using Gaussian elimination with time complexity $O(k^3)$. Per block, the decoding cost is thus quadratic.

Instead of Vandermonde matrices, Reed-Solomon erasure coding can also be implemented using Cauchy matrices [BKK+95]. The advantage of this method is that all of the arithemic operations can be converted into XOR-operations which are efficient to calculate on general-purpose computers.

Theoretically the best algorithms can decode Reed-Solomon codes with a complexity of $O(k \log^2 k \log \log k)$ [MS77]. However, these kinds of high speed theoretical algorithms are complex to implement and tend to have large hidden constants in asymptotic complexity, making them unsuitable for practical use. Additionally, faster algorithms often require more memory space than slower ones, resulting in time-space trade-off considerations. Thus, quadratic [Li05] or simpler log-quadratic [Did] time-decoding algorithms can be regarded as the best practical decoding strategies for Reed-Solomon erasure correction.

In short, the most serious drawbacks of using Reed-Solomon erasure coding are the limited number of possible symbols and the complexity of the encoding and the decoding algorithms. Although with recent hardware the coding can be implemented quite efficiently, resulting in a good encoding and decoding performance, we can do better and save computational effort and energy with modern coding techniques.

2.2 Fountain coding principle

The term *fountain coding* originates from the paper by John Byers et al. [BLMR98], where the authors present the *fountain coding principle*. An idealised digital fountain works as an infinite supply of packets, which are sent over the network to possibly multiple recipients. The original data are divided into k blocks (the source symbols) of equal size. The source then forms stochastically identical packets using these original blocks and, ideally, the possible number of such packets is infinite. The sender disseminates these packets throughout the network, and the receivers collect the packets which get through without errors. In an optimal situation, the receiver acquires exactly k of these packets, which can then be used to fully reconstruct the original data. The receiver does not care exactly which packets are received; any set of the sent packets is fine. Additionally, there is no need to use a feedback channel of any sort to acknowledge or ask for retransmissions, save the possible acknowledgement of the completion of the whole transmission. In contrast, for example, in TCP-based connection the received packets are required to be acknowledged. The connection times out if a certain packet takes too long to travel (e.g., if routed via a longer route for some reason) and results in retransmission and thus some wasted bandwidth. A transmission protocol employing the fountain principle does not require the book-keeping of outstanding packets, as the encoded packets could contain data from any location of the original file, and the packets

are not acknowledged.

A server spraying the packets works as a metaphorical fountain. The packets can be regarded as water drops and the recipient collects these drops into a bucket. A person collecting water from fountain does not care which specific water drops he gets; the only important quantity is the *amount* of water. When the bucket is full, the person is happy and finished with their water collection.

In practice, it is impossible to have a sender producing an infinite number of different packets. However, with k blocks it is possible to get an exponentially large number of different packets by, for example, using linear combinations of the original source blocks as the packets. In such a case the sender can produce 2^k different combinations as packets.

Further, the absolute minimum of k received packets is not enough for successful decoding with a high probability. When the packets are formed using a random algorithm some of the received packets are redundant, containing no new information for the receiver. For this reason we talk about *overhead factor* $f = 1 + \varepsilon$, where ε denotes the fraction of extra packets in addition to k needed to complete the decoding. Thus, the total number of packets the recipient needs for the successful decoding of the original message is $f \cdot k$.

The desirable properties for a protocol employing the fountain coding principle, or a fountain coding scheme in general, include (adapted from [BLM02] and augmented) the following:

- **Scalability**: the sender should be able to cope with any number of users.
- **Reliability**: there should be some guarantees that all recipients can decode an exact copy of the original data.
- **Low overhead**: ideally $f = 1$; in practice it should be as close to one as possible.
- **Time performance**: the encoding and decoding algorithms should have low time complexities.
- **Time independence**: a recipient should be able to start reception at any given time regardless of the other recipients.
- **Sender independence**: packets from any set of the possible senders should qualify for decoding.
- **Tolerance**: the whole process should tolerate different erasure probabilities and possible data rates for the receivers.
- **Ease of implementation**: to lower the threshold for actual implementation and widespread adoption, the methods should be conceptually easy.
- **Freedom from patents**: depending on the user and implementer, a patent-free solution might be preferable. The state-of-the-art fountain coding schemes are patented.

The idealised fountain scenario is not achieved by any existing coding method. There are trade-offs involved when choosing one coding method over another. For example, with Reed-Solomon codes, it is guaranteed that any k distinct received packets are enough for full decoding of the original data, and thus the overhead factor $f = 1$. However, the decoding complexity is high for large k and the number of possible packets is not as large as with many other codes. Further, with Reed-Solomon codes k and n need to be fixed beforehand, and n is limited by the Galois field used. In contrast, with true fountain codes, the encoded packets can be created on the fly without n being fixed in advance. Such codes are called *rateless* codes. With a rateless code, a potentially infinite number of distinct, stochastically equivalent descriptions of the source data can be generated. The code rate is not fixed before the encoding process begins.

The rateless property is ideal when the true channel conditions are not known in advance. All block codes need to have an estimate of the erasure probability p in order to choose an appropriate expansion factor n/k for the code. As the channel conditions are rarely static, the chosen parameters seldom stay optimal through the whole transfer process. Instead, with rateless codes the code rate needs not be fixed beforehand, and new packets are generated all the time until decoding is completed.

The origins of the basic idea of the digital fountain can be traced back to 1975, when N.F. Maxemchuk presented an idea of dispersity routing in his PhD thesis [Max75b] and in [Max75a]. Later, in 1989, M.O. Rabin presented an information dispersal algorithm [Rab89] for the reliable dispersal of data into multiple locations with a method closely resembling the fountain principle.

Last, a remark on the terminology used in this thesis is in order. In the context of fountain and related codes, the source symbols are the original message blocks and the encoding symbols are the packets. We will use the terms *block* and *packet* for the source and encoding symbol in most of this thesis to emphasise the focus on packet-level erasure correction.

2.3 LDPC, Tornado and random linear fountain codes

Universal fountain codes are asymptotically optimal for any erasure channel. This means that when the number of message blocks, k, grows to infinity the overhead needed tends asymptotically to the Shannon limit of the channel. Before discussing the first universal fountain codes, the LT codes, we briefly review the Tornado codes, which are considered to be the first fountain codes with a background in LDPC codes. We also review a fountain coding scheme based on sending random linear combinations of the source blocks.

2.3.1 LDPC codes

We briefly review the class of low-density parity-check (LDPC) codes, whose theoretical analysis and properties are ultimately behind the current success of the state-of-the-art fountain codes. LDPC codes were originally presented by Robert Gallager in [Gal62]. These codes can be used to correct bit errors, as well as erasures [PT03] [Pla05]. Although the codes

were discovered as early as in the 1960s, they were then largely forgotten as the computing power to utilise them fully did not exist. However, after their rediscovery [MN97] the codes attracted much attention from the coding theory research community. The LDPC codes have great asymptotic performance surpassing even that of turbo codes [BGT96], and code constructs achieving the Shannon capacity have been developed [LMS+97]. The driving factor behind the modern success of the LDPC codes is the progress in the random generation of the code structures and iterative decoding algorithms.

Let us consider a file which is divided into k distinct blocks of contiguous bits, where all of the blocks are of the same size. We denote $\boldsymbol{m} = (m_1, m_2, \ldots, m_k)$, where $\boldsymbol{m}$ is the file (message) and m_i the ith block. The LDPC codes are linear block codes, i.e., the encoding process generates linear combinations of the message blocks. The linear combinations are calculated over GF(2), i.e., binary alphabet [1], and the summation operation is XOR ($\oplus$). A generator matrix $\boldsymbol{G}$ specifies the relation between the source and encoding symbols.

The LDPC codes are *sparse-graph codes*, implying that the parity check matrix $\boldsymbol{H}$ corresponding to a certain LDPC code is sparse, meaning the matrix contains primarily zeroes as its elements. The codewords $\boldsymbol{x}$ of a certain code $\mathcal{C}$ have to satisfy

$$\boldsymbol{H}\boldsymbol{x} = 0 \ . \tag{2.3}$$

Gallager's original LDPC codes are *regular*. This means that for each row and each column of $\boldsymbol{H}$ the number of ones is the same.

The LDPC codes can be defined as codes on bipartite graphs. The two parts are formed by the message blocks and check nodes as presented in an example of a very small LDPC code in Figure 2.2 on the left-hand side. Another description can be given with the help of Tanner graphs, in which each node on the right-hand side does not directly present a data node, but represents a parity *constraint*, which enforces the sum of the connected message blocks to be zero. The parity check matrix corresponding to the graphs in Figure 2.2 is

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \ . \tag{2.4}$$

Several distinct types of LDPC codes can be defined. An irregular and systematic LDPC code is presented as an example in Figure 2.2. The original constructs by Gallager were not systematic; in the corresponding graphs this would be seen through the absence of k distinct degree-1 message blocks. LDPC codes can also be constructed by cascading several layers of check nodes.

The optimal decoding of LDPC codes is an NP-complete problem [Mac03]. Therefore, the academic studies on LDPC codes concentrate on finding such graph constructs which result in good performance when

[1] Using larger finite fields is also possible, but in this thesis we only address binary arithmetic.

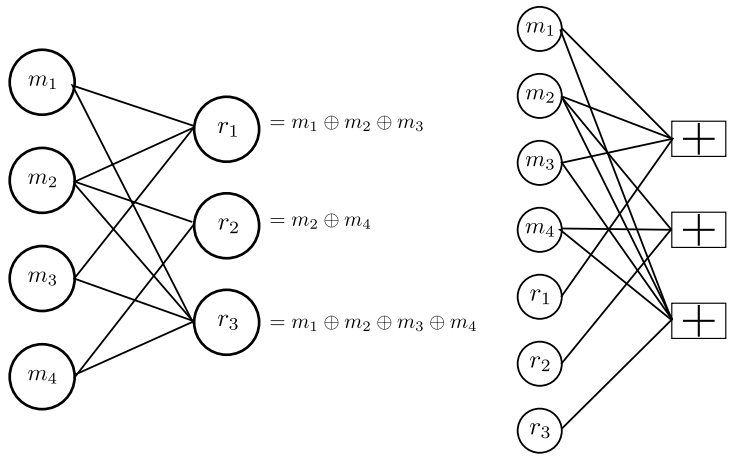


Figure 2.2: Two different representations of a systematic LDPC code as a bipartite graph. On the left-hand side the redundant nodes are presented as sums of the original message blocks. On the right-hand side is the representation of the code as a Tanner graph, where each of the nodes on the right is a constraint node forcing the parity of the neighbouring nodes to be zero.

decoded using iterative algorithms such as *message passing algorithm*. An efficient realisation of the message passing concept is *belief propagation algorithm*. In very general terms, such decoding algorithms work by sending *messages* or *beliefs* of their situation to neighbouring nodes in order to ultimately find an estimate $\widehat{\boldsymbol{x}}$, satisfying $H\widehat{\boldsymbol{x}} = \mathbf{0}$, for the original codeword $\boldsymbol{x}$.

When using LDPC codes over an erasure channel, one does not need to worry about possible erroneous bits, as the assumption is that those packets which get through are correct. The encoding symbols are packets which are linear combinations of the original source blocks, where the summation operation is bitwise XOR. This is a recurring theme throughout this thesis. The sent packets can be seen as linear equations conveying the information of exact contents, i.e., which blocks are XORred together, of each packet. The decoding of LDPC codes is simplified in an erasure correction setting; we analyse the decoding algorithm in more detail in the context of LT coding in Section 2.4.

Although the asymptotic properties of LDPC are of great interest, only little theoretical work has been done to analyse the finite-length scenarios. However, the LDPC codes have been shown also to be suitable for small code lengths, with considerably lower encoding and decoding complexity than Reed-Solomon codes. The reduction in the computational complexity is due to the iterative algorithms employed, which can be used because the LDPC codes are sparse.

More extensive treatments of LDPC codes and modern coding theory in general can be found, e.g., in [RU08] and [Mac03].

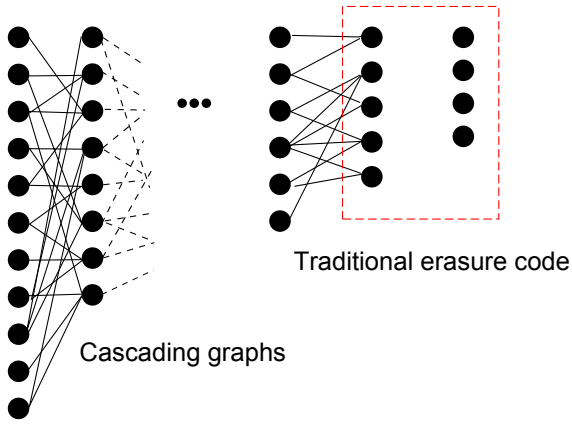


Figure 2.3: The original Tornado code is formed using cascading levels of coding and a traditional code on the last stage.

2.3.2 Tornado codes

In [BLMR98], Byers et al., besides discussing the fountain coding principle, also present Tornado codes as an example of an efficient erasure correcting code approximating the ideal digital fountain. An extensive analysis of this class of codes can be found in [LMSS01], while preliminary results appeared in [LMS+97],[LMA98] and [LMSS98].

Tornado codes are a capacity-achieving class of LDPC codes designed for erasure channels with a channel loss probability p. Tornado codes are capable of approximating the digital fountain concept better than Reed-Solomon codes [BLM02]. However, Tornado codes are not rateless. The rate is $R = 1 - p(1 + \varepsilon)$ (i.e., ε below the capacity of the erasure channel), and the time complexity of the decoding algorithm is proportional to $k \log(1/\varepsilon)\ \forall \varepsilon > 0$ per symbol. The key innovation made in [LMSS01] is the realisation that irregular codes perform better than regular ones. Especially with a large block length, the codes can operate with a low residual erasure probability arbitrarily close to the Shannon limit. Thus Tornado codes are *capacity-achieving* codes.

Tornado codes are systematic, and, in addition to the original message symbols, check symbols are generated and sent. As with the other code types presented later on in this thesis, the successful decoding of a Tornado code requires extra packets in addition to the k original blocks. Thus, the overhead factor $f > 1$. The original construct of Tornado codes uses cascading levels of LDPC coding to generate check symbols, i.e., the initial source blocks are encoded, the resulting encoding symbols are again encoded in a similar fashion, and so on. At the final level some traditional coding scheme, such as Reed-Solomon code, is used to generate the final check symbols (packets). Figure 2.3 depicts the idea. After the final stage there should be $O(\sqrt{k})$ check symbols. If the final code has quadratic decoding complexity (such as Reed-Solomon erasure codes), then the code constructed using cascading bipartite graphs can be encoded and decoded

in a period of time proportional to the total number of graph edges in the whole construct. This idea of using several layers of bipartite graphs to represent the code was presented for the first time in [LMSS01].

The graphs used in Tornado coding are formed using a random process, in which the degrees of the nodes are chosen from distributions which yield an irregular structure for the code. The analysis in [LMSS01] is done in terms of the degree sequences, i.e., the performance and capabilities of the code depend on said sequences.

Tornado decoding takes advantage of the fact that the code operates on an erasure channel. A missing (erased) message symbol can be decoded by setting it to be the XOR sum of all of the other message bits a neighbouring check bit is connected to. We present a more exhaustive treatment of this later in Section 2.4 in the context of LT coding, as the algorithm is exactly the same.

It was further shown in [RU01] that the coding can be performed without the cascading steps while retaining the linear encoding and decoding costs. Still, the graphs used in Tornado coding need to be explicitly constructed and n predefined, resulting in a finite code rate R. Compared to Reed-Solomon codes, Tornado codes sacrifice a little in their decoding inefficiency, as $f > 1$, but gain a lot in the time complexities of both encoding and decoding.

2.3.3 Random linear fountain

Good results in terms of the required overhead can be achieved by using random linear codes. We examine the random linear fountain (RLF), which is explained, e.g., in [Mac05]. The RLF is, as the name suggests, a random linear code approximating the ideal digital fountain principle.

A packet, i.e., an output symbol, c_i is generated by selecting each of the k message blocks to be included in a packet uniformly with probability $1/2$. This is equivalent to generating a random binary vector of length k. The packets are linear combinations of the selected blocks, where the blocks are summed using the bitwise XOR operation. Thus it is convenient to interpret each packet as a linear equation. The recipient collects these equations to form a linear system. When this system is of full rank with $n \geq k$ received equations (packets), the original file can be recovered. The recipient needs the information on which blocks are included in each packet in order to be able to decode the original data.

For example, consider a file of three blocks $\boldsymbol{m} = (m_1,\ m_2,\ m_3)$, from which the packets are generated. Assume that the receiver has successfully received the following set of equations:

$$\begin{cases} m_1 \oplus m_3 & = c_1 \\ m_1 \oplus m_2 \oplus m_3 & = c_2 \\ m_2 \oplus m_3 & = c_3 \end{cases}$$

which is a linear system of full rank. Thus, using the received packets c_1, c_2 and c_3 the recipient can decode the original file $\boldsymbol{m} = (c_2 \oplus c_3, c_1 \oplus c_2, c_1 \oplus c_2 \oplus c_3)$.

When the packets are generated as presented above, from the statistical

point of view the whole process is random as each packet is generated using the same information and a random algorithm. Thus no one packet is statistically more important than another, meeting one of the main ideal properties of fountain codes. Furthermore, this means that the loss process of the channel is irrelevant from the point of view of the process and its properties; the only thing that matters is the number packets received.

As already stated, the goal for the receiver is to receive k linearly independent packets successfully. Thus, we need to find the probability of a random binary matrix being of full rank. If the problem consists of k blocks, the number of different encoded packets is $2^k - 1$, excluding the all-zero packets, each representing an equation. Let us now consider how the number of linearly independent equations evolves from the recipients point of view. If we have one equation, the probability that we do not receive a linearly independent equation is

$$\mathcal{P}_1 = \frac{1}{2^k - 1} \ , \tag{2.5}$$

as we are unsuccessful only if we generate the same equation as before out of $2^k - 1$ different possibilities. In general, when we have m linearly independent vectors, the probability of failing to receive a new one is

$$\mathcal{P}_m = \frac{2^m - 1}{2^k - 1} \ , \tag{2.6}$$

where the numerator determines the size of the subspace of all linear combinations of m linearly independent vectors, again excluding the zero vector.

Let step m refer to the state in which m linearly independent equations have already been received. We use T_m to denote the random variable for the number of extra trials (in addition to the one we certainly need) needed for a new linearly independent equation at step m. It is geometrically distributed with the failure probability $\mathcal{P}_m$, that is,

$$\tilde{\mathcal{P}}_{m,i} = \mathcal{P}\left[T_m = i\right] = \mathcal{P}_m^i(1 - \mathcal{P}_m) \ , \quad i = 0, 1, 2, \ldots \tag{2.7}$$

The probability distribution for the total number of extra packets needed for successful decoding is then the same as the distribution of the sum of independent random variables $T = \sum_{m=1}^{k} T_m$:

$$\mathcal{P}\{T = i\} = \bigotimes_{m=0}^{k} \tilde{\boldsymbol{\mathcal{P}}}_m[i] \ , \tag{2.8}$$

where $\otimes$ denotes the convolution operation, and the square brackets denote the ith element of the resulting vector. The probabilities (2.8) depend mostly on the highest terms of $\mathcal{P}_m$, that is, on terms with m near k. As we can see from (2.6), the probability of unsuccessful generation depends almost only on $k - m$ for any reasonably sized k. The last step before k ($m = k - 1$) has a failure probability near to $1/2$, the second last $1/4$, and so on. This means that the actual convoluted probability distribution for the number of extra packets needed for decoding is already almost the

Table 2.1: The probabilities for the extra number of equations needed for a full rank linear system with k unknowns

$k + \#$ packets	probability (CDF)
0	0.289
1	0.578
2	0.770
3	0.880
4	0.939
5	0.970
6	0.984
7	0.992
8	0.996
9	0.998
10	0.999
11	1.000

same for $k \geq 13$ irrespective of the actual problem size k. The cumulative distribution function

$$\tilde{F}(t) = \mathcal{P}\left[T \leq t\right] = \sum_{i=0}^{t} \mathcal{P}\left[T = i\right] = \sum_{i=0}^{t} \left(\bigotimes_{m=0}^{k} \tilde{\mathcal{P}}_m[i] \right) \quad , \tag{2.9}$$

defines the probability that t extra packets (i.e., packets in addition to k) or less are sufficient for decoding. For $k \geq 13$, the distribution $\tilde{F}(t)$ is presented in Table 2.1.

The expected overhead for RLF can now be calculated with the help of Table 2.1:

$$\overline{T} = \sum_{i=0}^{\infty} (1 - \tilde{F}(i)) \approx 1.61 \quad . \tag{2.10}$$

This means that regardless of the size of the problem, the expected overhead of the random linear fountain is very low.

In Section 3.3 we need the distribution function $F(t)$ of the number of packets needed for successful decoding. This is simply related to that of the extra packets by

$$F(t) = \tilde{F}(t - k) \quad . \tag{2.11}$$

Further, let us use δ to denote the probability that decoding is not complete after receiving $n > k$ packets. It can be shown that $\delta \leq 2^{-n+k}$. The total number of received packets needed for succeeding in the decoding with probability $1 - \delta$ is $k + \log_2 1/\delta$, and we can get arbitrarily close to the Shannon capacity as $k \to \infty$ [Mac05].

Although the RLF has a very desirable overhead performance, the encoding and decoding costs make the scheme useful only for small message lengths k. The encoding cost is $k/2$ operations per packet on average, and the decoding requires Gaussian elimination, which is a $O(k^3)$ operation.

2.4 LT codes

Luby Transform (LT) codes [Lub02] are the first true universal fountain codes and a proper realisation of the fountain coding principle presented in Section 2.2. They are rateless and asymptotically optimal codes. Although they are nearly optimal for a large number of input blocks k, for a smaller number the overhead the LT coding process takes to provide a high probability of decoding is not close to the optimum. Optimisation for small lengths is discussed in Publications 1 and 2.

2.4.1 LT encoding and decoding

LT encoding and decoding algorithms use only the XOR operation between blocks and packets. The performance of LT codes is directly affected by the *degree distribution* used. A degree distribution $\boldsymbol{\rho}$ defines the probabilities $\rho(d)$ for choosing exactly d blocks for one packet. The degree refers to the degree of the node corresponding to the formed packet in the encoding graph. In the literature the symbol $\boldsymbol{\Omega}$ is also used to denote the degree distribution. Often, in the analysis of the asymptotic properties of fountain codes, a generator polynomial representation $\Omega(x) = \sum_{i=1}^{k} \Omega_i x^i$ is used.

The general LT encoding algorithm is described in Algorithm 1. In short, a packet is formed by first sampling a degree d from the degree distribution, then picking uniformly at random d blocks and calculating a linear combination of these using bitwise XOR. This process constitutes a bipartite graph, where the input nodes are the file blocks and the output nodes are the encoding packets, which are sent over the erasure channel. In order for the decoder to use iterative decoding on the graph, it needs to know the neighbours of every output node in the graph, i.e., the exact constituents of each packet. This information can be conveyed to the receiver, for example, by using headers on the sent packets or using a pseudo-random number generator initialised with a common seed at both ends of the channel. Which method is used is to be decided by the implementor to suit the specific scenario at hand.

The encoding cost of LT codes depends directly on the average packet degree. For one packet the encoding cost of calculating the linear combination is the packet's degree less one XOR operation. This average should be as low as possible in order to guarantee efficient operation, but also high enough to guarantee a high probability of decoding with as low an overhead as possible.

The definition of LT encoding that is presented is the one in [Lub02]. It is clear that in this form the LT codes are not systematic, unlike many traditional erasure correcting codes (Tornado codes, Reed-Solomon codes). However, there has been recent work on a construct of systematic LT codes [YP08], as will be discussed in more detail in the context of Raptor coding in Section 2.5. Another systematic approach using soft decoding is presented in [NYH07].

The decoding algorithm defined for LT coding is a special case of belief propagation decoding used for LDPC codes for erasure channels. Algorithm 2 provides an overview of a general LT decoding algorithm. The process starts with a degree-1 packet. As the recipient knows which par-

Algorithm 1 General LT encoding algorithm

1: **repeat**
2: choose a degree d from degree distribution $\rho(d)$.
3: choose uniformly at random d blocks $m(i_1), \ldots, m(i_d)$.
4: send $c \leftarrow m(i_1) \oplus m(i_2) \oplus \cdots \oplus m(i_d)$.
5: **until** enough output symbols are sent.

Algorithm 2 A general LT decoding algorithm

1: **repeat**
2: **while** no degree-1 packets in buffer $\mathcal{B}$ **do**
3: receive a packet and remove the known blocks.
4: store the packet in the buffer $\mathcal{B}$.
5: **end while**
6: $m(j) \leftarrow$ degree-1 packet from $\mathcal{B}$
7: **for all** $c \in \mathcal{B} : c$ includes $m(j)$ **do**
8: $c \leftarrow c \oplus m(j)$
9: **end for**
10: **until** original message is recovered.

ticular block is an exact copy of such a packet (i.e., is a neighbour in the bipartite graph), this immediately yields a new decoded block. This block is then subsequently XORred to any remaining neighbours in the graph, and the corresponding edges are removed. This removal reduces the degrees of the neighbouring packets, possibly revealing new degree-1 packets, which can subsequently be used to decode new blocks. This process is iteratively continued for as long as possible, until there are no degree-1 packets to process or the whole original message of k blocks has been decoded. An example of decoding for $k = 3$ is depicted in Figure 2.4. The topmost nodes correspond to the original blocks, while the nodes below are received packets.

The decoding algorithm is suboptimal in the sense that if degree-1 packets are absent, the process halts even though the relationships between the blocks and packets could in some cases be used to uniquely decode new blocks. As the packets are linear combinations of the message blocks, an algorithm such as Gaussian elimination could possibly be used for further decoding. However, the main reason for the success of LT codes is the time efficiency of the encoding and decoding processes. This means that costly operations such as Gaussian elimination should not be performed. Instead, the degree distribution design dictates the performance of both the encoding and decoding procedures. The remarkable result presented by Michael Luby in [Lub02] is that degree distributions which result in fast decoding and a low overhead do exist. These degree distributions further make it clear that the use of irregular graph constructs results in a very good performance and capacity-achieving codes.

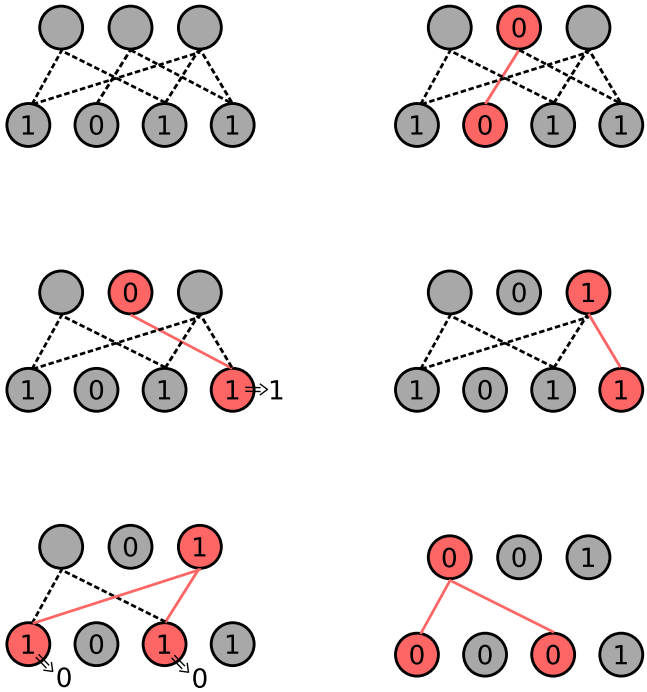


Figure 2.4: Decoding of LT codes for $k = 3$. The process starts with a degree-1 node (top right), which is XORred to the neighbouring block. Subsequently, the data in the block are XORred to connected packets and the edges removed (middle left). A degree-1 packet remains, which is used to continue the decoding (middle right). Ultimately, the two remaining degree-1 packets uniquely give the content of the last missing block. Processing either of these will complete the decoding.

2.4.2 Balls and bins and fountain coding

A simple degree distribution results by setting $\Omega_1 = 1$, i.e., all of the point probability mass is set for degree one. In [Lub02] this is called all-at-once distribution, hinting that this case is analogous to throwing balls into separate bins until each one of the bins contains at least one ball. Converted to our coding problem, this means that we want to cover all of the k message blocks with at least one packet, where the packets are formed by just selecting a random block and sending that one as a degree-1 packet.

Let us denote by A_i the event that bin i remains empty after n balls have been thrown into k distinct bins. Then we have

$$\mathcal{P}[A_i] = \left(1 - \frac{1}{k}\right)^n \to \left(\frac{1}{e}\right)^{\frac{n}{k}} \text{ as } n \to \infty \ . \tag{2.12}$$

Therefore, if we throw $n = -k \log \delta$ balls, the probability that a bin does not yet contain a ball is δ. We get a lower bound for the probability that all

of the bins contain at least one ball,

$$\mathcal{P}\left[\text{All bins contain at least one ball}\right] = 1 - \mathcal{P}\left[\bigcup_i A_i\right] \geq$$
$$1 - \sum_{i=1}^{k} \mathcal{P}\left[A_i\right] = 1 - \sum_{i=1}^{k} \left(1 - \frac{1}{k}\right)^n \to 1 - ke^{-n/k} \quad , \quad (2.13)$$

where we have used Boole's inequality. In other words, if we throw $n = k\log(k/\delta)$ balls into the bins we succeed in having at least one ball in every bin with a probability $> 1 - \delta$.

If there are i bins which already contain a ball, then the probability that a randomly thrown ball will drop into an empty bin is $(k-i)/k$. The number of throws needed for an empty bin to be occupied is clearly geometrical and the expected number of throws needed is $k/(k-i)$. Let us denote by X the random variable of the number of throws needed to cover all of the bins and by X_i the number of throws needed for an empty bin to be filled when i bins already contain a ball. Then $X = \sum_i X_i$ and by linearity of expectation we have:

$$E[X] = \sum_i E[X_i] = \sum_{i=0}^{k-1} \frac{k}{k-i} = k \sum_{i=0}^{k-1} \frac{1}{k-i}$$
$$= k\left(1 + \sum_{i=2}^{k} \frac{1}{i}\right) \leq k\left(1 + \int_1^k \frac{1}{x}\, dx\right) = k(1 + O(\log k)) \quad , \quad (2.14)$$

i.e., by sending approximately $k\log k$ degree-1 packets we can expect to have every message block covered.

2.4.3 Soliton distribution – ripple analysis

The LT process, which describes how the decoding of LT codes is performed, is defined in [Lub02]. The set of blocks which have been decoded but not yet subtracted from the packets they are neighbours of is called the *ripple*. In the LT process each block in the ripple is evaluated one at a time, subtracting it from all its neighbouring packets. If this step results in degree-1 packets, the corresponding blocks have been decoded and the ripple increases. The LT process ends when there is no further ripple to be processed.

In order to succeed in the decoding process, the ripple must not disappear until all of the blocks have been decoded. With a good degree distribution the ripple should be small enough not to release too many blocks at one time, to avoid an unnecessarily large ripple, and at the same time ensure that the decoding process keeps going on.

A probability distribution known as *soliton distribution* can be derived by walking through the LT process while keeping the expectation of the ripple equal to one throughout the decoding. This means that at every step, when time is increased by one, exactly one block is used to subtract one degree from its neighbouring packet and subsequently the packet is

processed. Additionally, the corresponding block is decoded, and thus the number of steps needed to decode the whole message is k.

Let us denote by $n(t, d)$ the random variable for the number of packets of degree d at the time instant t. The decoding process starts at time $t = 0$ when there is one degree-1 packet available, and none of the blocks has been decoded yet. At the beginning $E[n(0, d)] = k\rho_d$ and the total number of incident edges for degree d packets in the graph is $d \cdot \rho_d$. The ripple expectation condition can be stated as

$$E[n(t, 1)] = 1 \quad \forall t \in \{0, \ldots, k-1\} \ . \tag{2.15}$$

During the LT process, the expected number of packets whose degree is reduced from d to $d-1$ as a result of a block being processed is the expected number of edges incident to degree-d packets. This can be seen as follows: at time t, with $k - t$ blocks unprocessed (i.e., not yet decoded), the probability of a random degree-d packet being a neighbour to the processed block is $d/(k - t)$. When we multiply this probability by the expected number of degree-d packets, $E[n(t, d)]$, we get the number of packets whose degree is reduced. On the other hand, the expected number of edges incident to a packet of degree d is the total number of edges incident to the said packets divided by the number of unprocessed blocks:

$$\frac{d \cdot E[n(t, d)]}{k - t} \ . \tag{2.16}$$

The LT process is started with one degree-1 packet. To satisfy the condition $E[n(1, 1)] = 1$ one packet needs to reduce its degree from two to one in the transition from $t = 1$ to 2. That is, a degree-2 packet needs to be a neighbor to the block in the ripple at $t = 1$. We set the expected number of degree-2 packets whose degree is reduced to one, as this is sufficient to keep the process alive:

$$\frac{2E[n(t, 2)]}{k} = 1 \ , \tag{2.17}$$

that is, $E[n(t, 2)] = k/2$.

All of the nodes of degree d which are not neighbours to the block processed at time t remain as-is for the next step at $t + 1$. In addition, packets whose degree is $d + 1$ at time t and which are neighbours to this block will reduce their degrees by one, thus resulting in degree-d packets at time $t + 1$. Thus a general formula for the expected number of particular degree packets can be written in a recursive form as

$$E[n(t+1, d)] = \left(1 - \frac{d}{k-t}\right) E[n(t, d)] + \frac{d+1}{k-t} E[n(t, d+1)] \ . \tag{2.18}$$

A solution to this recursion is

$$E[n(t, d)] = \frac{k - t}{d(d-1)} \ , \tag{2.19}$$

with $n(0, 1) = 1$ as the initial value. The initial assumption in this analysis was that the decoder has $n = (1+\varepsilon)k$ received packets and does not receive any new ones during the decoding. Thus, the values we are looking for in order to derive a degree distribution are of the form $n(0, d)$. As $E[n(0, d)] = k\rho_d$, we get the *soliton distribution* by dividing by k.

Definition 2.1 (Soliton distribution) *The soliton distribution is*

$$\rho(d) = \begin{cases} \frac{1}{k} & \text{for} \quad d = 1 \ , \\ \frac{1}{d(d-1)} & \text{for} \quad d = 2, \ldots, k \ . \end{cases}$$

The average degree of soliton distribution is the harmonic sum $H(k) \approx \log k$.

In practice, soliton distribution performs poorly. The derivation was performed under the condition that the average ripple size is one, but in simulations of the LT-process with soliton distribution it can more often than not be seen that the ripple vanishes too early, resulting in a need for new packets to continue the decoding and possible unnecessary redundancy in the form of overlapping packets. Thus, the expected number of overhead packets needed is large, in addition to a large variance of the overhead. See Publication 2 and [Tir06] for some numerical evaluations.

2.4.4 Robust soliton distribution

As soliton distribution does not perform well enough to be used in practice as a result of the problem of a disappearing ripple, the distribution has been modified in a way to provide a large enough ripple for the LT decoding process to finish with a high probability. The resulting degree distribution is called the *robust soliton distribution* and is defined in [Lub02] as follows.

Definition 2.2 (Robust soliton distribution) *Let $R = c \cdot \log(k/\sigma) \cdot \sqrt{k}$, where $c > 0$ is a constant and δ is the probability of a decoding error after n received packets. Let*

$$\tau(i) = \begin{cases} R/ik & \text{for} \quad i = 1, \ldots k/R - 1 \ , \\ R \log(R/\sigma)/k & \text{for} \quad i = k/R \ , \\ 0 & \text{for} \quad i > k/R \ . \end{cases}$$

The robust soliton distribution is

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\sum_{i=1}^{k} \rho(i) + \tau(i)} \quad \text{for } i = 1, \ldots, k \ .$$

The motivation for adding $\tau(i)$ to the soliton distribution is to ensure that, on the average, the ripple is increased by one when a block is decoded in order to avoid the ripple disappearing completely. $\tau(1)$ tries to ensure a large enough ripple to begin the decoding, and the spike $\tau(k/R)$ tries to ensure that all of the blocks are eventually covered.

The average degree of the packets using robust soliton distribution is the same as with soliton distribution, $O(\log k)$.

In [Lub02] Luby presents a proof that when using robust soliton distribution, the probability that $k + O(\sqrt{2}\log^2(k/\delta))$ packets is not enough for decoding is $1 - \delta$. This is a remarkable result as it shows that there exist degree distributions which can be used with a simple XOR-based decoding algorithm to provide asymptotically optimal packet erasure coding.

The LT codes perform well when as regards the overhead, but the encoding and decoding time complexities can be made better. If we try to design an LT code with a constant encoding cost, the number of packets the receiver needs to collect is not close to k. This can be seen by proving (see the balls and bins in Section 2.4.2 and [Sho06]) that for LT coding for k blocks with a decoding algorithm, whose error probability is at most $1/k^c$ for some constant c, the bipartite graph corresponding to the code structure has $O(k \log k)$ edges. Therefore, if the decoding is to be completed with near to k collected packets, the cost of one encoded packet needs to be $O(\log k)$. In other words, if the degree distribution is designed to give a constant encoding cost, then inevitably the number of packets collected is not close to k. Thus, a fraction of the message blocks would remain undecoded. In other words, LT coding suffers from *error floor problem* [RU08].

2.5 Raptor codes

Raptor codes [Sho06] can be regarded as the current state-of-the-art form of fountain coding. They are actually an extension of LT codes, solving the innate error floor problem.

Basically, in Raptor coding a high-rate pre-code is used in addition to outer LT coding. The LT codes that are used are designed to have linear complexity, and the inner pre-code takes care of the error floor problem; that is, it alleviates the requirement that the LT code has to decode every one of the message blocks. Instead, a constant fraction is enough and the pre-code decodes all of the k message blocks.

The pre-code $\mathcal{C}$ is initially used to encode the k message blocks into *intermediate symbols*. These symbols are then used as an input for the LT encoder, which operates as required on the fountain principle generating a potentially endless stream of encoded packets. The pre-code and the degree distribution used $\mathbf{\Omega}$ need to be designed to ensure a high probability of decoding with $(1 - \varepsilon)k$ collected packets, where ε is a small constant. As the coding process involves two different codes, the encoding and decoding costs are not as easy to analyse as with LT codes. A thorough performance analysis of Raptor coding on binary erasure channels is presented in [Sho06].

The most basic examples of Raptor codes are the LT code, which is a Raptor code without the pre-code, and a pre-code only (PCO) Raptor code. In PCO Raptor code the degree distribution of the LT code is simply the all-at-once distribution discussed in Section 2.4.2. Thus, a PCO Raptor code can transfer any block code to a fountain code.

The application used dictates the choice of a good pre-code. Naturally, encoding and decoding costs should be as low as possible, and in addition the code should have a high rate. Suitable candidates are, for example, Tornado codes or high-rate Hamming codes. Further, the asymptotic properties of Raptor codes depend on the rate of the outer code; LT codes do not suffer from this kind of restriction.

Raptor codes can also be used as systematic codes. Systematic Raptor codes can be constructed by performing the encoding and decoding operations in what is, in a way, the wrong order. The idea is first to set k output

symbols, i.e., packets to correspond to the k original data blocks. Then, intermediate input symbols are derived by decoding the graph where the original blocks are the output symbols. Then an encoding algorithm is applied, and the resulting construct has the k original blocks as the first packets, followed by the required number of other Raptor-coded packets (repair packets). The graph (i.e., the degree distribution and the pre-code) has to be carefully designed in order for this method to work. A similar approach can be taken to derive the systematic form of LT coding [YP08].

Raptor codes are very efficient from the computational viewpoint, as the encoding and decoding costs per block/packet are both constant. A similar performance can be achieved with the codes known as Online codes [May02] and [MM03]. However, since the publication of these papers no further development seems to have taken place.

2.6 Applications

The original applications of fountain codes consider the reliable multicasting of large files to many recipients [BLMR98]. Since the publication of efficient fountain codes, they have been considered for use in the following applications.

- Underwater communications [CRZ08]
- Data dispersion in wireless sensor networks [AKS08]
- IPTV [LSW08]
- DVB-H [GBGC09]
- Internet protocols, such as FLUTE [PLL+04]
- Networked storage [DPR06]
- Deep-space communications [YCLX08]
- Delay-tolerant networking, DTN [AdP]
- In modern standards for sending media over cellular networks, 3GPP MBMS [3GP07]

This list is not comprehensive, but gives the idea of many possible applications gaining a benefit from using the fountain principle.

2.7 Summary

In this chapter we have presented the fountain coding principle and a brief history of fountain coding and the motivation behind it. Efficient packet erasure coding is mainly motivated by the need for a scalable transmission method for multicasting data to many recipients. Methods using acknowledgements and retransmissions are particularly inefficient in multicast scenarios because of possible feedback implosion at the sender's end.

Traditional erasure codes, such as Reed-Solomon codes, can be used to implement the digital fountain concept. These codes have the property

Table 2.2: Properties of presented coding methods. Typical encoding and decoding costs, the amount of symbols needed for decoding and additional notes are presented.

Code	Encoding	Decoding	Needed symbols, $f \cdot k$
Reed-Solomon[a]	$O(k)$	$O(k^3)$	k
Tornado[b]	$O(1)$	$O(k)$	$(1+\varepsilon)k$
Balls and bins	$O(1)$	$O(k)$	$O(k \log k)$
Random linear[c]	$O(k)$	$O(k^3)$	$k + O(1)$
LT	$O(\log k)$	$O(k \log k)$	$k + O(\sqrt{k} \log^2 k)$
Systematic LT	$O(1)$	$O(k)$	
Raptor	$O(1)$	$O(k)$	$(1+\varepsilon)k$
Online	$O(1)$	$O(k)$	$(1+\varepsilon)k$

[a] Uses finite field arithmetic. Theoretically faster decoding is possible, trade-off consists of high hidden asymptotic costs and complicated algorithms. The number of symbols n is limited by the size of the used finite field, this may result in more than k received packets needed. R-S codes are not rateless.

[b] Tornado codes are not rateless.

[c] See also Publication 3.

that for data of the size of k blocks, any k-set of collected unique encoding symbols is sufficient for decoding. However, the finite number of possible symbols and the computational complexity of the encoding and decoding algorithms result in only an approximation of the ideal fountain principle.

The LDPC codes can be used for erasure correction, when the sent packets consist of the original source blocks and check nodes, i.e., linear combinations of the original blocks. Efficient LDPC codes known as Tornado codes were the first published codes with linear time encoding and decoding algorithms to approximate the ideal fountain very efficiently. The trade-off in comparison with Reed-Solomon coding is inefficiency in the required number of received packets; the recipient needs $f \cdot k$, packets to decode the original data completely, where $f > 1$. With a good code design the decoding probability is high with f close to unity. Still, with Tornado codes, the total number of different possible encoding symbols n has to be determined beforehand.

A significant step forward was taken with the publication of LT codes, the first universal fountain codes which are rateless. The number of encoding symbols does not need to be determined beforehand, i.e., the packets are generated on the fly for as long as required. The performance of LT codes depends on the degree distribution, which has to be carefully designed in order to achieve low overhead characteristics. Constant time encoding for LT coding is not possible, as it suffers from the error floor problem. Raptor codes, an extension to LT codes using a pre-code before the application of an LT code eliminate this error floor problem and have constant-time encoding algorithms. A comparison of the properties of the modern fountain codes presented in this chapter is provided in Table 2.2.

A very low and constant overhead is achieved when each of the source

blocks is selected to be included in a packet with a probability 1/2. This idea is used in a random linear fountain (RLF). When the receiver has collected a linearly independent set of k packets, the decoding can be done using Gaussian elimination. The overhead when using RLF is only 1.6 packets on average, but the downside is the high computational complexity of the encoding and decoding algorithms.

Further issues include the need for some kind of flow control. The fountain principle itself is based on flooding to the network with packets, a scenario which not all network operators would probably like. Additionally, no killer application for fountain coding has yet been found. Although Raptor codes are part of some international standards, such as [3GP07], patent and other issues may result in slow adoption.

The research pace concerning different kinds of fountain coding methods has recently picked up some speed and an increasing number of research papers on the subject can be found. This chapter has presented the basics and scratched the surface of some advanced methods. In the following chapters we analyse and optimise different kinds of coding methods inspired by the fountain coding principle. LT coding in particular plays a large part in the methods described.

3 OPTIMISATION OF FOUNTAIN CODES

The performance of fountain coding and related packet erasure correcting schemes is determined by the ability to succeed in the data transfer process with an overhead that is as low as possible. In the literature the term *performance* is also often used when considering the time complexity of the different coding methods. In this thesis, however, by the performance of an erasure code we primarily mean the ability of the code to keep the wasting of bandwidth as small as possible. However, we will also discuss some aspects of the time complexity of the different schemes.

In [Lub02], LT codes are proved to be asymptotically optimal when robust soliton distribution is being used. This means that for a number of source blocks tending to infinity, the overhead needed for decoding is optimal, i.e., the overhead factor $f \to 1$ when $k \to \infty$. Indeed, for cases when the message length k is very large, preferably thousands or tens of thousands of blocks, robust soliton distribution with appropriate parameters works remarkably well in terms of low an overhead and low encoding and decoding complexity. Although probabilistic arguments can be used to prove the asymptotic optimality of LT codes, for small message lengths, and for any given k, the optimal degree distribution problem is still unsolved.

This chapter presents four aspects of the optimisation and development of erasure correcting codes based on the fountain principle for small message lengths. The first half deals directly with degree distribution optimisation for LT codes. In particular, we present an exact analysis of the system by modelling the LT coding process as a Markov chain and solve the optimal degree distributions for very small systems. We also present a combinatorial algorithm which can be used to derive results for up to 30 source blocks. Besides optimal degree distributions for toy cases, our results provide a general insight into the structure of good degree distributions. The problems with these methods are state space explosion and excessive computational complexity, which prohibit their use to cover longer file sizes.

For source files of hundreds of blocks we present an optimisation method based on simulations of the LT process. In our proposed optimisation strategy, an estimator for the performance measure is constructed using concepts from importance sampling theory. In particular, we use the concept of the change of the probability measure. This allows us to construct the estimator using simulation results generated with a specific degree distribution, and estimate the performance if another distribution was used. An iterative algorithm for optimising the degree distribution is proposed.

Random linear fountain (RLF), presented in Section 2.3.3, is an efficient fountain coding method in terms of low an overhead. The required overhead is lower than with LT coding for finite file lengths. The problem is the high computational cost of the encoding and, in particular, decoding algorithms for large k. A *divided random linear fountain* is a rateless code that is faithful to the fountain principle. The data are divided into segments of blocks and each of the segments is transmitted using RLF. The division is made to allow one to solve multiple parts which are of a more

computationally convenient size, compared to directly decoding the whole file. However, the division, with strict adherence to keeping all of the sent packets statistically equivalent, results in some inefficiency when a channel without feedback is used, as some packets are sent to parts of the problem that have already been decoded. Our contribution is a scheme in which LT coding is used on top of random linear fountain coding to alleviate this inefficiency. Using LT coding on top of RLF is further compared to using a data carousel method, in which packets are sent in sequence to the different parts until all the data have been decoded. Divided RLF can be used for file sizes of hundreds of source blocks.

We further proceed by presenting a novel sequential erasure correcting method. We sacrifice some of the ideal properties presented in Section 2.2 for the ideal fountain. In particular all of the file blocks are initially sent as-is, i.e., the code is systematic. Thus, the time-independence property no longer holds: the coding has a clear starting point and not all of the sent packet are stochastically identical. An advantage is that, unlike with LT coding, the first k packets contain only non-overlapping information. After the initial round repair packets are used to attempt to correct the possible gaps left by the erasure channel. The sender updates his belief about the situation of the receiver, based on the channel loss probability p, which is assumed to be known. Thus, additionally the channel independence property of an ideal fountain is lost. We optimise the sequence of the repair packet degrees using a greedy criterion based on the belief about the sender on the current state of the receiver. The code aims for simplicity and efficient performance attaining a low overhead of needed repair packets for code lengths up to a thousand.

This chapter is organised as follows. We describe our contributions to fountain coding optimisation in more detail in Section 3.1. The methods and results from Publications 1 and 2 considering the degree distribution optimisation of LT codes for small message lengths are summarised in Section 3.2. Section 3.3 presents the divided random linear fountain, studied in Publication 3, and the approaches that were studied to improve its efficiency. Finally we present the systematic, sequential erasure coding strategy in Section 3.4, based on Publication 4. The chapter is summarised in Section 3.5.

3.1 Contribution

In this section we describe our contribution in more detail. We start by considering the problem of finding optimal degree distributions for LT codes for small message lengths from a couple of message blocks up to hundreds of blocks in Section 3.2. First we present an exact Markovian analysis of the evolution of the state in the decoding process, in which a state corresponds to the set of received packets. A Markov chain is formed with state transitions and their probabilities determined by the probabilities of the reception of different possible packets. These probabilities are directly determined by the degree distribution used. The degree distribution is optimised by either minimising the expected number of steps in the chain from the initial state to an absorbing one (complete decoding), or maximising

the probability of the chain being in the absorbing state at the kth step. The degree distributions we find using this method are optimal; no better ones can be found.

The drawback of the Markovian model is that it can be analysed only for the toy cases $k = 3$ and $k = 4$, because of the state space explosion. We further present a combinatorial approach which uses a recursive algorithm to calculate optimal degree distributions for maximising the probability of decoding in k steps. This can be used to calculate results for up to $k = 30$. These two methods, with accompanying results, are studied in detail in Publication 1.

The small message length degree distribution studies are continued in Section 3.2.5. We present a method based on simulating the LT process and iteratively making the degree distribution better. We use a novel idea inspired by importance sampling techniques to find a function which approximates the performance for any degree distribution, not only for the one used in the simulations. In particular, this function allows us to use simulations to find an estimate for the gradient of the given performance measure with respect to the degree distribution. This gradient is then used at each step of an iterative algorithm to determine the direction of modification from the current degree distribution. Our numerical results suggest that the best performance is obtained with a degree distribution that has a spike similar to robust soliton distribution at some of the high degrees, and free parameters for degrees one and two. This method is studied in detail in Publication 2.

In the rest of this chapter we present two different erasure correction methods with some novel ideas. In Section 3.3 we study a scenario in which a random linear fountain (cf. Section 2.3.3) is divided into multiple parts and packets are generated from each of the parts using RLF. The division is made in order to make the parts reasonably fast to decode using a slow algorithm, i.e., we use the divide-and-conquer principle in the design of the encoding and decoding algorithm. The goal is to conserve the statistical identicality of the packets sent. This results in the selection of the part from which a packet is generated uniformly at random. However, when the operation is restricted to channels without feedback, the decoding of different parts is completed at different times without the sender knowing this; when the sent packets are statistically identical, some packets are sent to parts which are already decoded. Thus, with a random algorithm some efficiency is lost.

Our contribution is a method in which LT coding is used on top of the RLF encoded packets, generating what we call macropackets. The LT coding ensures that the macropackets are also statistically identical, and thus the rateless nature and independence from channel loss statistics are retained. This mode of operation alleviates some of the inefficiency resulting from dividing the original data into multiple parts of several blocks. We further compare the macropacket scheme with a data carousel-inspired algorithm, in which a packet from each of the parts is generated in turn until all the original data have been decoded. The methods and the research results are presented in detail in Publication 3.

In Section 3.4 we present a strategy in which we sacrifice the time-

independence property of ideal fountain coding and study how good a performance can be achieved in such a setting. The coding is started by sending each of the k original blocks as such; that is, the method is systematic. The time independence does not hold true any more as the sent packets are not statistically identical. As it is possible that during the initial systematic round some of the packets are erased, repair packets are generated by choosing a packet degree and sampling the original source blocks as in LT coding.

The repair packet degree optimisation works by allowing the sender to keep track of the sent packet degrees and update his belief about the state of the receiver. An estimate p of the channel repair probability is assumed to be known to the sender, and is used in belief updating. Basically, the belief is the probability distribution of the number of the packets the receiver is still lacking. This distribution is used in a greedy decision algorithm to determine the degree of the next repair packet. The repair packet degree selected is the one giving the highest probability of the packet immediately decoding a novel original block at the receiver's end. The optimal degree sequences are studied and discussed in detail identifying some common aspects and behaviours of the coding scheme. The encoding and decoding algorithms have low complexity and the overhead in the repair packets is low, $O(p)$. This novel systematic erasure code is studied in Publication 4. Although the code is not rateless in the same sense as LT coding, the number of possible repair packets is potentially infinite and does not have to be determined beforehand.

3.2 LT codes for small message lengths

Although using LT coding with very small lengths is not very practical, and arguably other erasure coding methods could provide even better performance [1], the exact analysis and results we present give us insight into the whole optimal degree distribution problem and reveal interesting things about the nature of good degree distributions.

First we will present a Markovian analysis of the LT process in Section 3.2.3. Using this model optimal degree distributions for $k = 3$ and $k = 4$ are presented. Then, in Section 3.2.4, we continue by a combinatorial approach, which can be used in optimisation for up to $k = 10$ analytically, and numerically for problem sizes of tens of message blocks. This presentation is a summary of Publication 1.

For somewhat larger k, hundreds of source blocks, we present the numerical optimisation algorithm based on simulating the LT process and constructing a target function for optimisation using concepts from importance sampling theory. Based on the simulation results we are able to calculate the direction of the gradient of the target function, i.e., we have the best direction to modify the degree distribution, allowing one to use iterative numerical methods for finding the optimal distribution. This method for optimisation of the degree distribution of LT codes is presented in Sec-

[1] For example Reed-Solomon erasure codes would require exactly k different received packets. The computational cost nor the limited number of possible symbols are not issues for the small lengths we are considering.

tion 3.2.5 and Publication 2.

3.2.1 Related research

The literature on optimising the degree distributions of LT codes for very small message lengths is scarce. In fact, at the time of writing this (June 2009) the author is not aware of any published article besides the ones included in this thesis, namely Publications 1 and 2, with a similar approach of directly finding the best possible degree distributions for small k. Some recent work on finite length analysis of LT coding is available, e.g., in [KLS04], but the employed methods are different from ours, and consider the error probability of the belief propagation algorithm.

Small length LDPC codes are studied for example in [PB05]. The used methods are combinatorial, and the studied message sizes fall under the same range studied in this thesis. Further, the resulting optimal overhead factors of small length LDPC codes are similar to small length cases of LT codes presented in this thesis.

The small length analysis and discussion in Publications 1 and 2 seems to have inspired research such as [BC08]. There the authors study robust soliton distribution and search for parameters giving good performance for message sizes $k = 100$ and less. Small message lengths are justified by real-time applications requiring low-latency in the transmission.

3.2.2 Optimisation objectives

The degree distributions for LT codes can be optimised using different criteria. Let us denote by T the number of steps (received packets) needed in the LT decoding process for completion (full decoding), and by Z_n the number of decoded blocks after receiving n packets and performing the possible iterative decoding steps. These are related by $T = \min\{n : Z_n = k\}$.

We define two different objectives for the coding optimisation problem:

1. Objective **Min.Avg.**: Minimise the expected number of steps $E[T]$ needed for full decoding.

2. Objective **Max.Pr.**: Maximise the probability of decoding with exactly k received packets. This is the point where full decoding is possible for the first time. We denote this probability by $\mathcal{P}_k = \mathcal{P}[Z_k = k] = \mathcal{P}[T = k]$.

These two optimisation goals are closely related and result in only slightly different degree distributions when the optimisation results are compared.

The second optimisation goal, **Max.Pr.**, is close to those defined in the literature, in which the studies typically consider the failure probability of the belief propagation decoder.

3.2.3 Markovian model

We study the LT process presented in Section 2.4 as a Markov chain. The set of distinct packets the receiver has collected determines the state of the chain. The decoder performs the decoding in a particular state as far as it can, possibly revealing fully decoded blocks. The process starts when the

receiver has no packets at all. The degree distribution used in the encoding process determines the probabilities of different packets. The form of the received packet, in turn, determines the corresponding state transition.

Let us consider the case with $k = 3$ message blocks and denote the original blocks by a, b and c. If the receiver has already decoded the first block, a, in the message and has received a packet with second and third block summed together, bc, the system would be in the state $\{a, bc\}$. The LT process ends when all of the blocks are decoded, that is, the state $\{a, b, c\}$ is the absorbing state.

The total number of non-zero packets the sender can combine out of k blocks is $2^k - 1$. As each of these potential packets either belongs or does not belong to a particular state denotation, the total number of possible states is 2^{2^k-1}, resulting in very high growth in the problem size with k.

The states formed by considering all possible combinations of distinct received packets are called the *raw states* of the system. We reduce the number of these raw states by *decoding reductions*, *isomorphism reductions* and *state aggregation*.

By decoding reductions we mean situations in which the iterative decoder can be used to simplify the state denotation. For example, state $\{a, abc\}$ can be reduced to the state $\{a, bc\}$ by subtracting the first decoded block from the packet abc.

Further reduction in the number of raw states can be achieved by considering isomorphisms. Any two states that can be derived from each other by permutation of the blocks in the state denotation are isomorphic. Each set of isomorphic states can be represented by a canonical representative, which can be determined by ordering the isomorphic states in some unique way and picking up the first state in the list as the canonical representative[2]. Thus the total number of states needed to be considered can be significantly reduced from the original raw states. An example of a set of isomorphic states is: $\{\{a, bc\}, \{b, ac\}, \{c, ab\}\}$, which all can be represented by the canonical representative $\{a, bc\}$. Essentially, all of these states consist of one degree-1 block and the other two possible blocks XORred together. Thus the structure of the example states is the same.

After doing the two basic forms of state space reduction a final step of state aggregation can be performed by analyzing the states more closely. An example of the states and the transitions for $k = 3$ is depicted in Figure 3.1. When we examine the four states in the shaded box before the absorption state on the right side, i.e. $\{ab, bc\}, \{ab, ac, bc\}, \{ab, ac, abc\}$ and $\{ab, ac, bc, abc\}$, we see that any received degree-1 packet in any of these states will result in complete decoding and transition to $\{a, b, c\}$. As a result, these four states can be aggregated into one macro state.

Using the above reductions, we can reduce the number of states in the chain for toy cases of $k = 3$ and $k = 4$ from 128 and 32768 raw states, respectively, to 9 and 87 states. However, for $k = 5$ the number of reduced states remains as 161065, which is too high from computational point of view.

[2]The reason why it is sufficient to consider a single permutation of the blocks is that in the encoding process the data blocks are sampled uniformly at random once the packet degree is chosen.

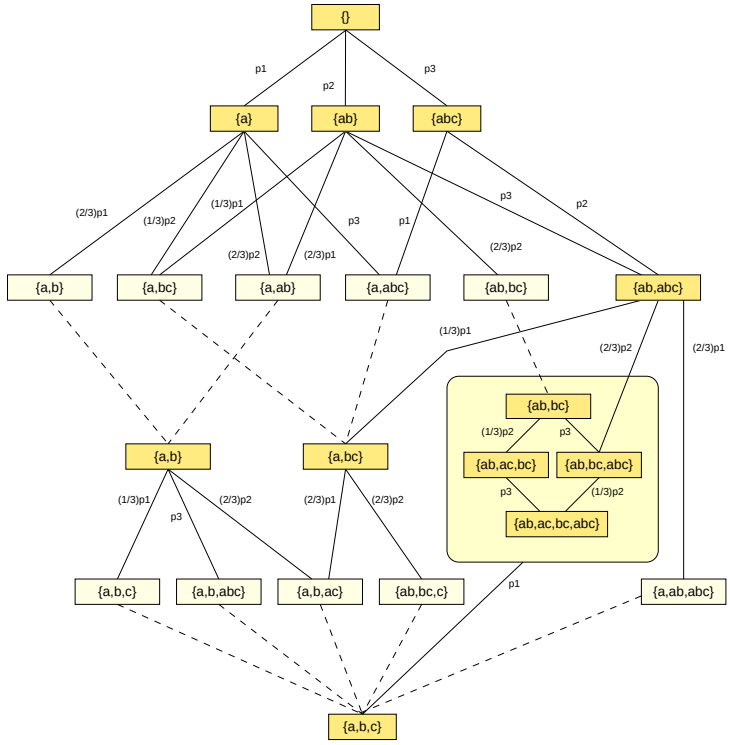


Figure 3.1: Example of state transition diagram for $k = 3$. The dark shaded boxes represent the reduced states before state aggregation. Solid lines represent state transitions, the probabilities are given corresponding to the components of the degree distribution. Dotted lines represent reductions of raw states. The four states in a rounded box in the right side can be further aggregated into one state, resulting in 9 uniquely different states in the system.

Probabilities p_{ij}, with which the state of the Markov chain changes from i to j, define an $N \times N$ state transition matrix $\boldsymbol{P}$, where N is the number of reduced states. The state probabilities p_{ij} depend on the used degree distribution, $\rho(d)$. The state transition matrix $\boldsymbol{P}$ is generated by considering the reduced state sets. The idea is to consider all possible arriving packets to each of the states, subsequently decoding what is possible and replacing the resulting state by its canonical representative. The probability of the degree of the used packet is then added to the corresponding entry in state transition matrix $\boldsymbol{P}$. This algorithm is listed in Algorithm 3.

Using the state representations as described above the optimisation problem can be formulated as follows. We partition the state transition matrix $\boldsymbol{P}$ in a canonical form:

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} , \tag{3.1}$$

where $\boldsymbol{Q}$ is the transition matrix between transient states, $\boldsymbol{R}$ represents tran-

Algorithm 3 Generation of the transition matrix $\boldsymbol{P}$

1: generate all possible 2^{2^k-1} raw states
2: remove all states that are reducible by decoding and retain only the canonical representative of each class of permutation isomorphic states
3: **repeat**
4: take a state s from the reduced state space
5: **repeat**
6: generate an arriving packet $pckt$, i.e., a non-empty subset (combination) of the n blocks
7: add $pckt$ to the set of packets in the intial state s, reduce the new state by decoding and finally find its canonical isomorphic representative s'
8: add $\rho(d)$ to the entry ss' of the transition matrix, where d is the degree of $pckt$
9: **until** all distinct packets are generated
10: **until** all states of the state space are gone through

sitions from transient states to absorbing ones and $\boldsymbol{I}$ is identity matrix corresponding to the absorbing states. In our case, we have only a single absorbing state, i.e., the state in which the decoding is fully completed. Algorithm 3 can initially be used to form $\boldsymbol{P}$, and if necessary the states can be relabelled so that the transient states come first, resulting in the canonical form (3.1).

For an absorbing chain, the fundamental matrix $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{Q})^{-1}$ is positive and well-defined. An element $m_{ij} \in \boldsymbol{M}$ gives the expected number of times the chain visits state j when the chain starts in state i. Using the fundamental matrix, it is possible to calculate the expected number of the transitions needed for decoding:

$$\begin{aligned} E[T] &= \boldsymbol{\pi}_0 \boldsymbol{M} \boldsymbol{e}^{\mathbf{T}} \\ &= \boldsymbol{\pi}_0 (\boldsymbol{I} - \boldsymbol{Q})^{-1} \boldsymbol{e}^{\mathrm{T}} \ , \end{aligned} \tag{3.2}$$

where $\boldsymbol{\pi}_0 = (1,\ 0,\ \ldots,\ 0)$ is the initial distribution vector corresponding to an empty system, and $\boldsymbol{e}^{\mathrm{T}} = (1, \ldots, 1)^{\mathrm{T}}$. For objective **Min.Avg**, we can directly optimise this quantity.

For objective **Max.Pr.** we need to consider the probability of full decoding on exactly k steps. Using the canonical form (3.1) we can write this probability (actually, the probability of decoding with any number of steps) as:

$$\mathcal{P}_k = \boldsymbol{\pi}_0 \boldsymbol{P}^k \boldsymbol{\pi}_{\mathrm{abs}}^{\mathrm{T}} \ , \tag{3.3}$$

where $\boldsymbol{\pi}_{\mathrm{abs}} = (0\ \ldots\ 0\ 1)$ corresponds to the absorbing state.

For $k = 3$ we optimise the degree distribution in the sense of the two objectives discussed above. The Markov chain is constructed for a system with the reduced state space consisting of 9 states. Using Algorithm 3, we

Table 3.1: Optimal weights in case $k = 3$

	Min.Avg.	**Max.Pr.**	binomial	soliton	uniform	deg-1
p_1	0.524	0.517	3/7	2/6	1/3	1
p_2	0.366	0.397	3/7	3/6	1/3	0
p_3	0.109	0.086	1/7	1/6	1/3	0
$E[T]$	4.046	4.049	4.133	4.459	4.725	5.5
$\mathcal{P}_3$	0.451	0.452	0.437	0.397	0.354	0.222

can find the transition probability matrix

$$\boldsymbol{P} = \begin{pmatrix}
0 & p_1 & p_2 & p_3 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{p_1}{3} & 0 & 0 & \frac{2p_1+2p_2}{3} & \frac{p_2}{3}+p_3 & 0 & 0 & 0 \\
0 & 0 & \frac{p_2}{3} & 0 & \frac{2}{3}p_1 & \frac{p_1}{3} & p_3 & \frac{2}{3}p_2 & 0 \\
0 & 0 & 0 & p_3 & 0 & p_1 & p_2 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{2p_1+p_2}{3} & 0 & 0 & 0 & \frac{p_1+2p_2+3p_3}{3} \\
0 & 0 & 0 & 0 & 0 & \frac{p_1+p_2+3p_3}{3} & 0 & 0 & \frac{2p_1+2p_2}{3} \\
0 & 0 & 0 & 0 & 0 & \frac{p_1}{3} & \frac{p_2}{3}+p_3 & \frac{2}{3}p_2 & \frac{2}{3}p_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & p_2+p_3 & p_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix},$$

where $p_i = \rho(i)$. Using (3.2) we now have explicitly

$$E[T] = \boldsymbol{\pi}_0(\boldsymbol{I}-\boldsymbol{Q})^{-1}\boldsymbol{e}^{\mathrm{T}} = \frac{1}{p_1}+p_1\cdot\left(\frac{6}{p_1-3}+\frac{18}{(3-p_2)(3-2p_1-p_2)}+\frac{9}{2(p_1+p_2)(3p_1+2p_2)}\right). \tag{3.4}$$

From this result it is easy to calculate the optimal weights if we want to have as few steps as possible on average to decode the message. Similarly, using (3.3) it is easy to find the optimal weights that maximise the probability of full decoding after 3 received encoded packets.

For $k = 4$ similar kinds of constructs are used to optimise the degree distributions. A computer algebra system (Mathematica [Wol]) is used to automatically generate the state transition matrix for the 87 reduced states and optimise the corresponding symbolical forms of (3.2) and (3.3).

The optimisation results for both the **Min.Avg.** and **Max.Pr.** criteria are listed in Table 3.1 for $k = 3$ and in Table 3.2 for $k = 4$. In addition to the optimisation results, the performance with binomial, soliton, uniform and all-at-once distributions (deg-1) are presented. Both uniform and all-at-once distributions perform rather poorly, the all-at-once distribution being worst. Using only degree-1 packets suffers from the problem of filling the last few of the missing blocks. In contrast, the binomial distribution performs reasonably well.

The difference in performance between the two objectives is insignificant. In particular, the distribution that is optimal in the **Max.Pr.** sense works very well also for the **Min.Avg.** criterion. Also, the ranking of the other three distributions is the same as before. Binomial distribution works rather well with respect to both criteria (though, in relative terms, not quite as well as in the case $k = 3$), while the other two are much worse.

Table 3.2: Optimal weights in the case $k = 4$

	MinAvg	*MaxPr*	binomial	soliton	uniform	deg-1
p_1	0.442	0.429	4/15	3/12	1/4	1
p_2	0.385	0.430	6/15	6/12	1/4	0
p_3	0.112	0.100	4/15	2/12	1/4	0
p_4	0.061	0.041	1/15	1/12	1/4	0
$E[T]$	5.580	5.590	6.255	6.276	7.182	8.333
$\mathcal{P}_4$	0.314	0.315	0.257	0.262	0.184	0.094

3.2.4 Combinatorial approach

For the **Max.Pr.** objective, i.e., maximising the probability $\mathcal{P}_k$ of decoding in exactly k steps, we can use an alternative, recursive approach, which is viable for larger values of k. We take advantage of the fact that the order of the received packets is irrelevant; with a specific set of k received packets the decoding problem can either be solved fully or then some of the original data blocks remain undecoded.

In order to emphasise the dependence of $\mathcal{P}_k$ on the degree distribution, let us write $\mathcal{P}_k = P_k(p_1, \cdots, p_k)$, where $p_i = \rho(i)$ are the point probabilities of the degree distribution $\rho(d)$. We calculate this on the condition that of k received packets $k - m$ have degree 1 and are all distinct. The first of the conditioning events happens with $(k - m)$th point probability of the $\mathrm{Bin}(k, p_1)$ distribution. For the decoding to start, the LT decoder needs at least one degree-1 packet, and thus $(k-m) \geq 1$. Duplicate packets are not allowed, as in the optimal situation all of the received data have to be novel to the receiver. The probability of all of the $k - m$ degree-1 packets being mutually distinct is

$$\frac{k}{k} \cdot \frac{k-1}{k} \cdot \cdots \cdot \frac{m+1}{k} = \frac{(k-1)!}{m!\, k^{k-m-1}} \quad . \tag{3.5}$$

Thus, by conditioning on the number of degree-1 packets we have

$$\begin{aligned} P_k(p_1, \ldots, p_k) \;&=\; \sum_{m=0}^{k-1} \binom{k}{m} p_1^{k-m} (1-p_1)^m \\ &\quad \times \frac{(k-1)!}{m!\, k^{k-m-1}}\, P_m(p_1^{(k,m)}, \ldots, p_m^{(k,m)}) \quad , \end{aligned} \tag{3.6}$$

where

$$p_j^{(k,m)} = \sum_{i=2}^{n} \frac{p_i}{1-p_1} \frac{\binom{m}{j}\binom{k-m}{i-j}}{\binom{k}{i}} \quad , \quad j = 1, \ldots, m \tag{3.7}$$

is the degree distribution of the remaining m packets originally of degree of at least 2, after the degree-1 packets are subtracted. Fraction $p_i/(1-p_1)$ is the probability of a packet to be of degree i given that it is not a degree-1 packet. The remaining fraction gives the probability of the event that when i original blocks are chosen from the k possible, we choose exactly j blocks from the m still undecoded blocks. That is, the reduced degree of the packet is j after subtraction of the degree-1.

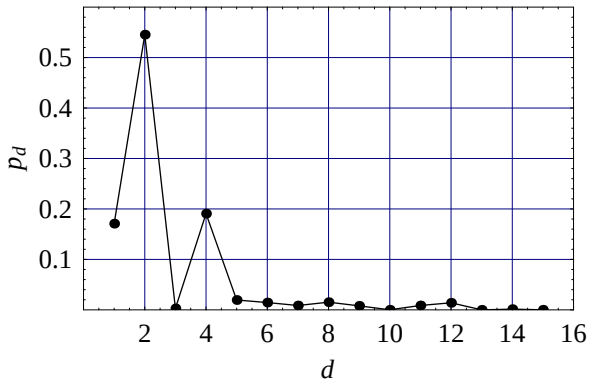


Figure 3.2: Optimised degree distribution for $k = 15$ using the recursive combinatorial approach.

The recursion (3.6) can be solved and simplified to give

$$\begin{aligned}
\mathcal{P}_0 &= 1 \\
\mathcal{P}_1 &= P_1(p_1) = p_1 \\
\mathcal{P}_2 &= \frac{1}{2}p_1^2 + 2p_1p_2 \\
\mathcal{P}_3 &= \frac{2}{9}p_1^3 + \frac{4}{3}p_1^2p_2 + 2p_1p_2^2 + 2p_1^2p_3 + 4p_1p_2p_3 \\
&\vdots\ ,
\end{aligned} \tag{3.8}$$

where $\mathcal{P}_0$ and $\mathcal{P}_1$ are the seeds for the recursion.

For small k, the recursion and optimisation can be made in symbolic form using a suitable computer algebra system (such as Mathematica [Wol]). Examples of optimisation results are given in Table 3.3. In addition, the symbolic form can be used for consistency checking of the results obtained using the Markov chain optimisation model presented in Section 3.2.3. The results are exactly the same as for cases $k = 3$ and $k = 4$ using the Max.Pr. objective. For larger k the number of terms in (3.8) becomes soon unmanageable for symbolic computations. For example, for $k = 10$ the number of terms is already 16796.

Results for larger k can still be obtained by numerically calculating the recursion (3.6). This can be done for up to $k = 30$ in approximately one minute on a modern desktop PC. For optimisation purposes we can use values such as $k = 20$, for which the computation of one point takes less than a second. An example of optimised degree distribution for $k = 15$ is presented in Figure 3.2. More results and discussion on the optimal form of the degree distribution are presented in Section 3.2.6.

3.2.5 Simulation based optimisation

Importance sampling is a method often used in conjunction with Monte Carlo integration methods to reduce the variance of the estimates of inter-

Table 3.3: Optimal weights for **Max.Pr.** objective in cases $k = 5, \ldots, 8$

k	5	6	7	8
p_1	0.370	0.327	0.294	0.268
p_2	0.451	0.467	0.480	0.491
p_3	0.102	0.099	0.093	0.085
p_4	0.055	0.068	0.082	0.099
p_5	0.021	0.024	0.021	0.013
p_6		0.014	0.020	0.027
p_7			0.009	0.010
p_8				0.007
$\mathcal{P}_k$	0.226	0.166	0.124	0.094

est. Variance reduction makes it possible to reduce the number of samples needed for estimation with a certain accuracy. The basic idea is to use another distribution to make important rare events in the simulation more frequent, and then compensate for this by an appropriate weighting factor, called the *importance ratio* [Rub97] [Ros00].

Instead of using the importance sampling as a variance reduction technique, we use this technique to estimate the value of the objective function for a different degree distribution than what was originally used in the simulations. In particular, an estimate for the gradient of the estimator with respect to the degree distribution is constructed, and the best direction to improve the distribution is obtained. This technique is then used in a numerical iterative algorithm to find the optimal degree distribution.

The proposed method is based on simulations. One simulation run is a full realisation of the complete LT encoding and decoding process for a specific size message using a degree distribution given, with the point probabilities $p_i = \rho(i)$. We generate a set of samples $S_k = \{R_k, \boldsymbol{n}^{(k)}\}$, where R_k is the outcome of the kth simulation run, i.e., the value of the objective function in that run, and $\boldsymbol{n}^{(k)}$ is a vector where component $n_i^{(k)}$ gives the number of degree-i packets generated and sent during the simulation run.

We use the two optimisation goals, **Min.Avg.** and **Max.Pr.** defined in Section 3.2.2. For **Min.Avg.** we minimise $E[R] = E[T]$ and for **Max.Pr.** we maximise $E[R] = E[\mathbf{1}(T = k)]$.

Before introducing the estimate of R, we briefly review the principle of importance sampling. The expectation of function h of a continuous random variable $\boldsymbol{X}$ with probability density function $p(\boldsymbol{X})$ is

$$E[h(\boldsymbol{X})] = \int h(\boldsymbol{x}) p(\boldsymbol{x}) \, d\boldsymbol{x} \ . \tag{3.9}$$

This can also be written as

$$E[h(\boldsymbol{X})] = \int h(\boldsymbol{x}) \frac{p(\boldsymbol{x})}{g(\boldsymbol{x})} g(\boldsymbol{x}) \, d\boldsymbol{x} \ , \tag{3.10}$$

where $g(\boldsymbol{x}) \neq 0 \forall \boldsymbol{x} : p(\boldsymbol{x}) h(\boldsymbol{x}) \neq 0$. Now, if m samples $\{\tilde{\boldsymbol{X}}_{i=1}^{m}\}$ are drawn

from $g(\boldsymbol{x})$, an unbiased estimate of the expectation can be calculated as

$$\hat{h} = \frac{1}{m} \sum_{i=1}^{m} w(\tilde{\boldsymbol{X}}^{(i)}) h(\tilde{\boldsymbol{X}}^{(i)}) \ , \tag{3.11}$$

where

$$w(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \tag{3.12}$$

is the *importance ratio*.

The simulation samples $S_k, k = 1, \ldots, m$ are used in constructing the estimate of R. The importance sampling idea is used here: Simulations are performed using a given degree distribution $p_i, i = 1, \ldots, k$, but the value of the estimator is obtained for any other degree distribution $q_i, i = 1, \ldots, k$:

$$\hat{R}(\boldsymbol{q}) = \frac{1}{m} \sum_{k=1}^{m} R_k \prod_i \left(\frac{q_i}{p_i} \right)^{n_i^{(k)}} \ . \tag{3.13}$$

Thus, (3.13) gives an estimate of R if distribution $\boldsymbol{q}$ is used and, in principle, allows to an optimisation to be performed with respect to $\boldsymbol{q}$. However, there is a numerical problem, as the estimate has a large variance with a finite number of samples, especially when $\boldsymbol{q}$ and $\boldsymbol{p}$ are far from each other. Further, as the number of free parameters in the distribution grows, the accuracy becomes worse and more simulation samples are required for the estimator to be formed.

For a practical optimisation algorithm, (3.13) is used to derive another estimate, namely for the *gradient vector* of the objective function with respect to the degree distribution. The ith component of the gradient of the estimate $\hat{R}$ with respect to $\boldsymbol{q}$ is

$$\hat{g}_i = \frac{\partial \hat{R}}{\partial q_i} = \frac{1}{m} \sum_{k=1}^{m} R_k n_i^{(k)} \frac{1}{p_i} \left(\frac{q_i}{p_i} \right)^{n_i^{(k)} - 1} \ . \tag{3.14}$$

We evaluate the gradient at $\boldsymbol{q} = \boldsymbol{p}$, to give a direction in which the candidate for a better degree distribution would lay:

$$\left(\frac{\partial \widehat{R}}{\partial q_i} \right)_{\boldsymbol{q}=\boldsymbol{p}} = \frac{1}{m} \sum_{k=1}^{m} R_k \frac{n_i^{(k)}}{p_i} \ . \tag{3.15}$$

Thus, by simulating with degree distribution $\boldsymbol{p}$, an estimate for the direction of the gradient is obtained.

When the gradient is known, it can be used in a numerical algorithm to find the optimum (minimum or maximum, depending on the goal) of the estimator. In Publication 2 we have used the gradient method with bisection line search [BSS93] to take the step towards a better distribution. In addition to knowing the direction of the step, it needs to made sure that the degree distribution resulting from taking the step is feasible. This can be carried out by projecting it to hyperplane $\boldsymbol{g} \cdot \boldsymbol{e} = 0$, and constraint the step size so that all of the resulting components $(p_{\text{new}})_i \in [0, 1]$. The simulation is continued until a (pre-defined) stopping criterion is met. We

have calculated the sample standard deviation of the estimate $\widehat{R}$ and stop the algorithm when a certain threshold is met.

For details of the optimisation algorithm, see Publication 2. We give a simplified summary of the algorithm here. The form presented here works with point distributions, in which the probabilities p_i are directly optimised. It is easy to extend the algorithm for parametrised distributions, the details are presented in Publication 2:

1. Use the given probability distribution (either point or parametrised form) as a starting distribution $\boldsymbol{p}$.

2. Generate samples S_k using the degree distribution $\boldsymbol{p}$. Generate samples until accuracy is less than ε.

3. Use the samples S_k to calculate the gradient $\boldsymbol{g}$. This is, in the case of point distribution, the projection of the gradient vector.

4. Do a bisection search in the direction of the gradient.That is, optimise $\widehat{R}(\boldsymbol{p} + \lambda \boldsymbol{g})$, by finding either a minimum or maximum depending on the goal. As a result we have the step length λ.

5. The step towards a better distribution is: step $\leftarrow \lambda \cdot \boldsymbol{g}$.

6. Limit the change of each component to 90% of the previous value. This percentage can also be varied. This ensures that the conditions $\sum p_i = 1$ and $p_i \in [0, 1]$ are met for point distributions.

7. Move in the direction of the gradient: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \text{step}$.

8. Calculate the value of $\widehat{R}$ and the standard deviation $\sigma_{\widehat{R}}$. If the absolute difference between the last two estimates is less than the standard deviation, then stop. Otherwise continue and go back to step 2.

The novel idea of using the importance sampling-inspired gradient estimator works reasonably well when the number of samples is large and the number of free parameters to be optimised is low. When all of the individual point probabilities are regarded as free parameters, optimisation is efficient only for up to $k = 10$, after which the number of required samples is too high leading to computational and memory issues. Further, the optimisation goal **Max.Pr.** behaves better than **Min.Avg.**. When maximising the decoding probability, line search converges well to a clear maximum, and an implementation is easy to make. However, this is not so when minimising the average number of steps needed. The main problems are choosing a suitable interval for the line search and the fact that $\boldsymbol{q} = 0$ is a global minimum (cf. (3.13)). Thus, the direction of the gradient and the convergence of the algorithm need to be carefully controlled to produce good results.

In Publication 2, equations (3.13)-(3.15) are also presented in a parametrised form, allowing the use of degree distributions in which only a couple of parameters dictate the form of the distribution. Numerical evaluations indicate that the best results, i.e., the degree distributions giving the lowest overhead, have only a couple of components which need to be optimised.

Table 3.4: Results from 10000 runs of LT simulations for $k = 100$

Distribution	Average T	Std $\hat{\sigma}(T)$
(3.16)	125.0	13.1
(3.16) with spike at $i = 50$	123.9	9.9
ideal soliton	169.5	72
robust soliton, $\sigma = 0.5, c = 0.01$	148.5	44.8
robust soliton, $\sigma = 0.5, c = 0.03$	134.9	23.9
robust soliton, $\sigma = 0.5, c = 0.1$	132.9	13.3

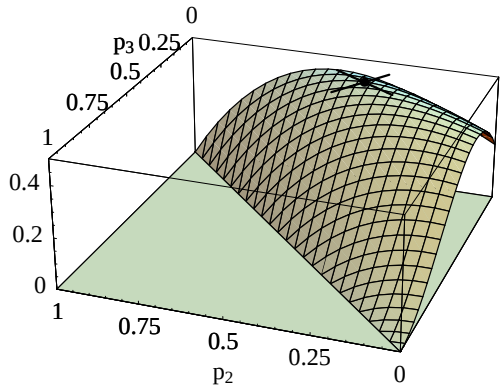


Figure 3.3: $\mathcal{P}_3$ as function of p_2 and p_3. The ridge is steep in one direction and less so in others.

The most effective form, with three parameters η_1, η_2 and η_3 for $k = 100$ is found to be

$$p_i = \begin{cases} \eta_1, & \text{for} \quad i = 1 \ , \\ \eta_2, & \text{for} \quad i = 2 \ , \\ \eta_3, & \text{for} \quad i = 50 \ , \\ \frac{1}{i(i-1)}, & \text{for} \quad i = 3, \ldots, k, i \neq 50 \ , \end{cases} \tag{3.16}$$

which is actually soliton distribution with the first two degrees as free parameters, with an added "spike" at $i = 50$. This form mimics that of robust soliton distribution, in which the spike is also present. Note that the distribution needs to be normalised, and thus the form (3.16) has only two free parameters. Numerical results from simulations when using the optimised parameters in (3.16) are presented in Table 3.4.

3.2.6 Results – The form of optimal degree distribution

The results presented in both Publications 1 and 2 suggest that a degree distribution has only a few parameters that have to be correctly adjusted to get a good performance. Also, the best form of degree distributions remains

similar to that of the robust soliton distribution, with a 'probability spike' on some higher degree. This can be seen, for example, from the optimal form in Figure 3.2, in which the combinatorial approach is used, and from the results in Table 3.4. The robust soliton distribution is optimal in the asymptotic case $k \to \infty$, but degree distributions of the form resembling it seem to perform well also in the finite length cases, even for small k.

The insensitivity of the performance to the degree distributions can be further studied by calculating the eigenvectors and -values of the second derivative matrices $\partial^2 \mathcal{P}_k / \partial p_i \partial p_j$ at the optimum (maximum) point using the analytical forms of $\mathcal{P}_k$, presented in Section 3.2.4. The eigenvectors indicate the principal directions, and with the corresponding eigenvalues the magnitude of the curvatures in these principal directions can be studied. Figure 3.3 depicts $\mathcal{P}_3$, with principal directions displayed as small arrows. The two eigenvalues corresponding to the eigenvectors in this case are $\lambda_1 = -6.97$ and $\lambda_2 = -0.71$. In the direction of the second principal direction, changing the degree distribution would result in a very slight change in the performance of the coding.

In general, the eigenvalues of the principal directions form a decreasing sequence, in which the typical ratio between two consecutive eigenvalues is of the order of 10. For example, when $k = 10$ the largest eigenvalue (in magnitude) is $\lambda_1 = -27.3$ while the lowest is $\lambda_9 = -2.53 \cdot 10^{-6}$, indicating that the function is highly insensitive to changes in the direction of the last eigenvector. The conclusion from these considerations is, that the performance is very insensitive to changes in many directions, and all distributions represented by points laying at the intersection of a few hyperplanes perform well.

3.3 Random linear fountain – divide and conquer

As explained in Section 2.3.1 the fountain codes can be perceived as sending linear equations over the erasure channel. Random linear fountain (RLF) can be used to approximate the ideal digital fountain. RLF operates by sending randomly selected linear combinations of source blocks. Random selection in this context means selecting each of the original blocks with probability $1/2$ and calculating the linear combination of the selected blocks using the exclusive-or operation, as explained in Section 2.3.3.

The main drawback of using RLF is that the computational complexity of decoding grows very fast with the number of source blocks, as the corresponding linear system is dense, and not a sparse one generally preferred in fountain coding. Dense systems typically need to be solved with high complexity algorithms, such as the Gaussian elimination algorithm with time complexity of $O(k^3)$. An advantage of the RLF scheme is a very low, and constant, average overhead of 1.6 packets, in contrast to much higher overheads for finite length cases with Tornado, LT and Raptor codes.

We analyse divided random linear fountain in Publication 3. Our main driving factor for the division lays in the growing computational complexity of RLF with the message size. Therefore, the problem is divided into multiple parts of smaller size. To preserve the statistical equivalence of all of the generated packets, the coding is performed by first picking the part

uniformly at random. After a part is picked, a packet is generated and sent using RLF as presented in Section 2.3.3. Information on the constituent blocks of the RLF packet and the index of the part the packet is from are sent to the recipient, e.g., in the packet header, along with the packet itself. Each of the parts works as their own RLF and the receiver can continue with the decoding only by receiving packets from the corresponding part. The encoding procedure is depicted in Figure 3.4.

The purpose is to study how the division affects the overhead and what measures can be taken to overcome the possible degradation in the performance. The sender does not know if a particular part is already solved, but keeps on choosing the parts from which to generate RLF encoded packets at random. This will eventually result in sending superfluous packets to those parts already solved, especially when there are only one or a couple of undecoded parts left. To make the performance degradation lower, we study how clumping many of the RLF produced packets together using LT coding, a method known as *macropackets*, works. We compare this to a *data carousel*-inspired scheme, which works by selecting different parts for packet generation in sequence. The intent with macropackets is again to preserve the statistical equivalence of the sent packets, as the LT encoding works by selecting the blocks (here the parts of the whole problem) at random. However, the data carousel does not have this property; the packets have sequence numbers, and therefore are not statistically identical.

We use a slightly different notation in this section compared to the rest of the thesis, stemming from the original notation in Publication 3. We denote by k the number of parts and by n the number of source blocks in each of the parts so that $N = k \cdot n$ is the total size of the source data. This thesis uses the word *part* when referring to the segments of multiple original file blocks resulting from the division. In Publication 3 these parts are referred to as *subproblems*.

3.3.1 Related research

The division scheme and the possible performance degradation is similar what would happen when, for example, Reed-Solomon coding is used for large data sets in which the number of finite field symbols is not large enough [BLM02].

The data carousel has been studied in the context of many different communications problems, in particular, e.g., in multicasting and broadcasting scenarios.

3.3.2 Divided random linear fountain

The encoding procedure of divided RLF is depicted in Figure 3.4. A part is chosen uniformly at random, and a packet is generated by sampling the blocks in the selected part with probability $1/2$ and finally combining them with the bitwise XOR operation. The division of RLF into k parts would ideally result in a total overhead of $k \cdot 1.6$ packets, if each of the parts would work as a separate RLF. The minimal overhead can be achieved only if the sending of the repair packets to each of the parts is stopped immediately when the corresponding part of the data are decoded. If we follow the

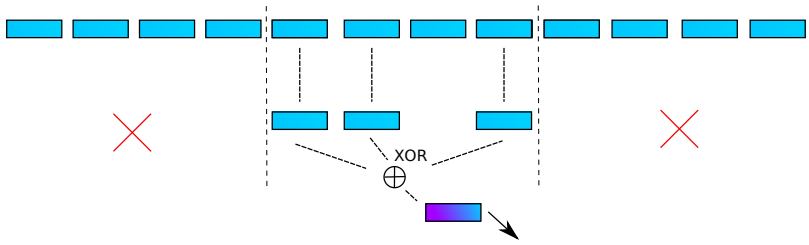
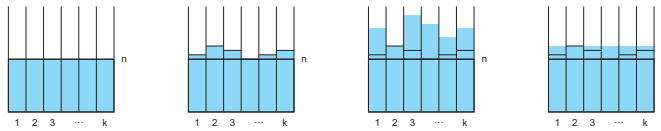


Figure 3.4: Illustration on how the encoding works in divided random linear fountain. A part is selected uniformly at random and a packet is generated by choosing each of the blocks with probability 1/2.



(a) Ideal situation, n packets for each of the parts

(b) Separate acknowledgements for each part

(c) Packets sent to randomly chosen parts

(d) Data carousel with low channel loss rate

Figure 3.5: Illustration of the decoding process for different strategies of selecting the part in which a packet is generated in. The shaded areas indicate the total number of packets received and lines the number of needed packets.

fountain coding paradigm, such information, however, is not available to the sender as the backward channel does not exist or is not used[3]. Thus, some of the packets might be sent to the already decoded parts of the original data resulting in an increased overhead.

The amount of overhead required for different modes of operation of divided RLF is illustrated in Figure 3.5. The height of the bars correspond to the number of sent packets for each of the k parts shown as distinct bins. The horizontal lines represent the ideal case of n received packets and the number of packets ultimately required for a full decoding of the whole data. The ideal situation, in which exactly n packets are enough, is presented on the far left in Figure 3.5a. The middle left shows the scenario in which each of the parts is acknowledged upon completion, resulting on average in $1.6 \cdot k$ total overhead packets. The scenario in which the packets are sent randomly to any of the parts is presented on the middle right (this is our divided RLF scenario). The packets may be sent into parts already completed as long as there is one part still lacking packets. This results in inefficiency, seen as the bar above the upper horizontal line in Figure 3.5c. The last picture, furthest on the right, shows the scenario when data carousel is used

[3]Or it may be used only for the acknowledgement of the decoding of the whole message

to send an equal number of packets to each of the parts. The process is completed when the last part is decoded, effectively resulting in the same amount of overhead in all of the parts.

To support our reasoning we present an exact analysis of the performance of divided RLF. Let ν be the total number of received packets and let further $\boldsymbol{\nu} = (\nu_1, \nu_2, \ldots, \nu_k)$, where ν_i denotes the number of received packets into part i, i.e. $\nu = \sum_i^k \nu_i$. The encoding algorithm selects the part at random, thus the probability of a generated packet belonging to part i is $1/k$. Then $\boldsymbol{\nu}$ obeys the multinomial distribution:

$$\mathcal{Q}(\boldsymbol{\nu}) = \mathcal{P}(\boldsymbol{\nu} = (\nu_1, \ldots, \nu_k)) = \frac{\nu!}{\nu_1!\nu_2! \cdots \nu_k!} \left(\frac{1}{k}\right)^{\nu} = \nu! \prod_{i=1}^{k} \frac{\left(\frac{1}{k}\right)^{\nu_i}}{\nu_i!} \quad . \tag{3.17}$$

Given the composition of $\boldsymbol{\nu}$, the probability of successful decoding of all the parts is $F_n(\nu_1)F_n(\nu_2)\cdots F_n(\nu_k)$, where F_n is the distribution function of the number of packets needed for successful decoding of n source blocks in the random linear fountain, given by (2.11). By deconditioning, the probability $\mathcal{R}_k$ of successful decoding with at most ν received packets is

$$\mathcal{R}_k(\nu) = \nu! \sum_{|\boldsymbol{\nu}|=\nu} \prod_{\kappa=1}^{k} \frac{\left(\frac{1}{k}\right)^{\nu_\kappa}}{\nu_\kappa!} F_n(\nu_\kappa) = \nu! \sum_{|\boldsymbol{\nu}|=\nu} \prod_{\kappa=1}^{k} G(\nu_\kappa) \quad , \tag{3.18}$$

where $G(\nu) = \frac{(1/k)^\nu}{\nu!} F_n(\nu)$. Now, the sum is recognised to be of the type that can be calculated by convolution:

$$\mathcal{R}_k(\nu) = \nu! \bigotimes_{\kappa=1}^{k} \boldsymbol{G}[\nu] \quad , \tag{3.19}$$

where $\boldsymbol{G} = (G(0), G(1), \ldots, G(\nu))$.

In Table 3.5 we have listed some performance results of divided RLF calculated using (3.19). The results show that with an increasing number k of the parts the performance deteriorates rapidly. These results support the reasoning above, and are in accord with the fact that the expectation of the minimum of k i.i.d. random variables is less than the mean, the difference growing with k.

In the following, methods which can be used to alleviate the performance loss, associated with the division of the original data set into multiple parts, are studied. The next section introduces the concept of macropackets and the exact performance analysis of scenarios with $k = 2$ and $k = 3$. After this the data carousel, which makes the number of packets in each of the parts identical, is considered.

3.3.3 Macropackets

From the discussion above, it should be clear that the strategy of choosing the part from which RLF packet is generated at random makes the average overhead too large. Especially when many of the parts are already solved, most of the sent packets are wasted and the decoding of the last few parts may take considerable amount of time. Thus, we propose using

Table 3.5: Mean number of packets required for decoding and corresponding overhead percentages for divided random linear fountain with k parts and n blocks per part.

N	k	n	E$[T]$	overhead %
128	2	64	141	10
	4	32	160	25
	8	16	196	53
	16	8	268	109
200	2	100	215	7.5
300	2	150	317	5.7
	3	100	331	10
400	2	200	419	4.8
450	3	150	487	7.6
600	3	200	641	6.8

Algorithm 4 The encoding algorithm with macropackets

Require: a file divided into k parts each of n blocks, macropacket degree distribution $\rho(d)$
  repeat
    choose macropacket degree d from $\rho(d)$
    choose uniformly at random d parts $M(i_1), M(i_2), \ldots, M(i_d)$
    for each chosen $M(i_h)$ **do**
      $c_h \leftarrow$ empty, the to-be RLF encoded packet
      for $j \leftarrow 1$ to n **do**
        choose and set $c_h \leftarrow c_h \oplus m_j$ in $M(i_h)$ with probability $1/2$
      end for
      store c_h
    end for
    send macropacket $c_1 \oplus c_2 \oplus \cdots \oplus c_d$
  until enough packets are sent

macropackets, which are generated using an LT code over all of the parts. A macropacket degree d is first sampled from the degree distribution $\rho(d)$, d parts are then chosen uniformly at random, and a packet is generated by using the RLF encoding algorithm in each of these selected parts. These packets are finally XORred together to form the macropacket. The algorithm for the macropacket generation is listed in Algorithm 4 and depicted in Figure 3.6.

The macropackets help especially when only one of the parts is still lacking packets to achieve full decoding. When generated over multiple parts, the content of the macropacket corresponding to the already resolved parts can be subtracted, in effect reducing the macropacket into a packet from the one remaining part. As stated when discussing the LT coding in Chapter 2, the information on which parts are included needs to be included in the macropacket in one way or another. Thus, the LT decoder works through the received macropackets possibly revealing degree-1 macropack-

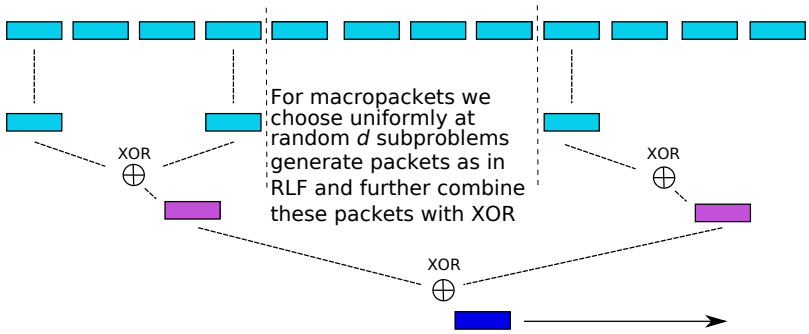


Figure 3.6: A macropacket is generated by choosing d parts of the original data, generating packets using RLF in each of those parts and calculating a linear combination of the generated packets.

ets, which then can be fed into the corresponding part for the RLF decoder. When all of the parts have enough packets, the RLF decoder in each part is able to recover all of the original data blocks.

When the original data is divided into $k = 2$ or $k = 3$ parts an exact analysis of the performance of divided RLF with macropackets can be made, by taking into account all the possible ways to decode a part. We will present an analysis of the case $k = 3$.

There are 7 different kinds of macropackets possible for $k = 3$. Let us denote these by A, B, C, AB, BC, AC and ABC, corresponding to which of the parts A, B and C are chosen to be included in the macropacket. Let $\boldsymbol{\nu}$ denote the vector specifying the number of received macropackets of each kind.

The decoding probability with $\nu = |\boldsymbol{\nu}|$ received packets is

$$\mathcal{R}_k(\nu) = \sum_{|\boldsymbol{\nu}|=\nu} \mathcal{Q}_k(\boldsymbol{\nu})\mathcal{P}_k(\boldsymbol{\nu}) \ , \tag{3.20}$$

where $\mathcal{Q}_k(\boldsymbol{\nu})$ is the multinomial distribution giving the probability of a specific macropacket (cf. (3.17)) and $\mathcal{P}_k(\boldsymbol{\nu})$ is the probability of decoding with the specified $\boldsymbol{\nu}$.

For $k = 3$ we need to derive $\mathcal{P}_3$ in (3.20). Taking the symmetry into account cases of three different types are identified: (i) either all of the parts are solved directly by degree-1 macropackets, i.e., as with standard divided RLF, (ii) two of the parts are solved directly with divided RLF and one with help of the macropackets and (iii) one part solved directly with two others solved with help of macropackets. Let $A[\nu], B[\nu]$ and $C[\nu]$ denote the events that part A, B or C is solved when ν macropackets of corresponding type are available. Further, $\mathcal{P}\left[A[\nu]\right] = F_n(\nu)$, where F_n is the distribution function of the number of packets needed for successful decoding of n source blocks in the random linear fountain, see Section 2.3.3. Using this notation, we have for (i)

$$\mathcal{P}(A[\nu_A] \cap B[\nu_B] \cap C[\nu_C]) = F(\nu_A)F(\nu_B)F(\nu_C) \tag{3.21}$$

and for (ii)

$$\mathcal{P}(A[\nu_A] \cap B[\nu_B] \cap C[\nu_C + \nu_{AC} + \nu_{BC} + \nu_{ABC}] \cap \overline{C[\nu_C]}) \\ = F(\nu_A)F(\nu_B)(F(\nu_C + \nu_{AC} + \nu_{BC} + \nu_{ABC}) - F(\nu_C)) \ , \quad (3.22)$$

allowing for decoding with the help of macropackets of type AC, BC and ABC but not for direct solving of C. By symmetry we have similar equations for solving A or B with help of macropackets.

Case (iii) is a little bit trickier. We take as an example the case where A is solved directly but the others are not. We have three different possibilities for solving B and C:

1. B and C are solved with macropackets common with A corresponding to the event

$$A[\nu_A] \cap B[\nu_A + \nu_{AB}] \cap \overline{B[\nu_B]} \ .$$

2. First B is solved with macropackets common with A and then C with macropackets common with B, corresponding to the event

$$A[\nu_A] \cap B[\nu_A + \nu_{AB}] \cap \overline{B[\nu_B]} \cap C[\nu_C + \nu_{AC} + \nu_{BC} + \nu_{ABC}] \cap \\ \cap \overline{C[\nu_C + \nu_{AC}]} \ .$$

3. First C is solved with macropackets common with A and then B with macropackets common with C, corresponding to the event

$$A[\nu_A] \cap C[\nu_A + \nu_{AC}] \cap \overline{C[\nu_C]} \cap B[\nu_B + \nu_{AB} + \nu_{BC} + \nu_{ABC}] \cap \\ \cap \overline{B[\nu_B + \nu_{AB}]} \ .$$

We add up the probabilities of all these three possibilities and arrive at

$$\mathcal{P} = F(\nu_A)((F(\nu_B + \nu_{AB}) - F(\nu_B))(F(\nu_C + \nu_{AC}) - F(\nu_C)) \\ + (F(\nu_B + \nu_{AB}) - F(\nu_B))(F(\nu_C + \nu_{AC} + \nu_{BC} + \nu_{ABC}) - F(\nu_C + \nu_{AC})) \\ + (F(\nu_C + \nu_{AC}) - F(\nu_C))(F(\nu_B + \nu_{AB} + \nu_{BC} + \nu_{ABC}) - F(\nu_B + \nu_{AB}))) \ . \quad (3.23)$$

Again, because of the symmetry, the total probability of a specific composition of macropackets requires adding up three of these equations which we get by cyclically permuting A, B and C in (3.23).

Now, using (3.20) we can write the total decoding probability, where

$$\mathcal{P}_3(\nu_A, \nu_B, \nu_C, \nu_{AB}, \nu_{BC}, \nu_{ABC}) = F(\nu_A)F(\nu_B)F(\nu_C) \\ + F(\nu_A)F(\nu_B)(F(\nu_C + \nu_{AC} + \nu_{BC} + \nu_{ABC}) - F(\nu_C)) + \dots \\ + F(\nu_A)((F(\nu_B + \nu_{AB}) - F(\nu_B))(F(\nu_C + \nu_{AC}) - F(\nu_C)) \\ + (F(\nu_B + \nu_{AB}) - F(\nu_B))(F(\nu_C + \nu_{AC} + \nu_{BC} + \nu_{ABC}) \\ - F(\nu_C + \nu_{AC})) + (F(\nu_C + \nu_{AC}) - F(\nu_C))(F(\nu_B + \nu_{AB} + \nu_{BC} + \nu_{ABC}) \\ - F(\nu_B + \nu_{AB}))) + \dots \ , \quad (3.24)$$

where in the place of dots one should substitute cyclic permutations of A, B and C in (3.22) and (3.23). The analysis of $k = 2$ subproblems can be performed in similar way and is presented in Publication 3. These results can be used to perform an exact performance analysis of divided RLF using macropackets.

3.3.4 Data carousel

Instead of using an LT code as an outer code to divided RLF, the number of wasted packets can be reduced by using the encoder on each of the parts in sequential order instead of selecting the part in random. This kind of operation mode can be seen as a special case of a *data carousel*. A data carousel operates by sending data blocks sequentially one after another over a communications channel, starting from the beginning again when the last block is sent. Eventually the recipient should get all of the blocks. The drawback is that the performance of this kind of operation is often poor if the channel is lossy.

We use the packet erasure channel model as introduced in 2.1.1. This means independent losses with probability p. As explained in Publication 3, the performance can be calculated using the decoding probability $\mathcal{P}_k(\boldsymbol{\nu}) = F_n(\nu_1)F_n(\nu_2)\cdots F_n(\nu_k)$. The sampling of $\boldsymbol{\nu}$ can be carried out by using the fact that the difference of the sequence number of two consecutive received packets is geometrically distributed. Samples from a geometric distribution give the part number by calculating $(\sum_{j=1}^{i} U_j) \mod k$, where the U_j are realisations of a geometric random variable. This way we can count the number of packets belonging to each of the parts out of total ν, and get the components of $\boldsymbol{\nu}$. For details we refer to Publication 3.

3.3.5 Numerical results

Before moving into the performance results obtained using macropackets and the data carousel scheme, we take a look at Table 3.5 from Publication 3, which shows the mean number of packets required decoding with basic divided RLF (cf. Figure 3.5c). The performance degradation is severe when the data is divided into more than two or three parts. The results additionally show reduction in the overhead percentage as the original problem size N grows. This is in line with the fact that the scheme is asymptotically optimal.

This degradation can be reduced by either using the macropackets or the data carousel-inspired scheme. The data carousel makes the number of the sent packets to each of the parts the same. As discussed above, however, the data carousel violates the strict fountain principle as the sent packets are not statistically identical.

Now, when using the macropackets we calculate the performance as presented in Section 3.3.3. Already for $k = 3$, evaluation of (3.20) would require six nested summations, which effectively implies that a direct direct calculation of R_3 is impossible for any practical values of n and k. Thus, the generation of results is performed using Monte Carlo summation in (3.20).

Essentially this translates to

$$\mathcal{R}_k(\boldsymbol{\nu}) \approx \frac{1}{S} \sum_{s=1}^{S} \mathcal{P}_k(\boldsymbol{\nu}_s) \ , \tag{3.25}$$

where the samples $\boldsymbol{\nu}_s$ are drawn from the multinomial distribution $\mathcal{Q}_k$ and S is the sample size.

Sampling from the multinomial distribution can be performed by sampling its marginal distributions, i.e., binomial distributions. The idea is to first sample the number of type-1 packets from $\mathrm{Bin}(\nu, p(1))$ and continue by subtracting the number of already generated packets from ν and using conditioning:

$$\begin{cases} \nu_1 \sim \mathrm{Bin}\left(\nu, p(1)\right) \ , \\ \nu_i \sim \mathrm{Bin}\left(\nu - \sum_{j=1}^{i-1} \nu_j, \ \frac{p(i)}{\sum_{j=i}^{k} p(j)}\right) \ , \quad i = 2, 3 \ldots \\ \nu_k = \nu - \sum_{i=1}^{k-1} \nu_i \ . \end{cases} \tag{3.26}$$

Also, the performance of data carousel can be calculated using (3.25), as described in Section 3.3.4.

Numerical results for macropackets are presented in Figure 3.8 and Table 3.6. An example of the performance of data carousel with different channel loss probabilities p is plotted in Figure 3.7. Macropackets and the data carousel scheme are further compared with the scenario in which separate acknowledgements are sent for each completed part, and packets are generated uniformly from each of the parts. The degree distributions for macropackets were numerically optimised for best results.

As already discussed in Section 3.3.2, the performance of the divided random linear fountain is rather poor. The worst case performance (i.e., when the decoding success probability $\to 1$) results in an overhead percentage of 50% for $n = 32$ and 25% for $n = 100$. If a backward channel and separate acknowledgements were utilised, the optimum performance of $3 \cdot 1.6 = 4.8$ overhead packets on average could be achieved. The performance of the macropacket scheme falls approximately in the middle between these two extremes. An interesting feature is that the data carousel for $n = 32$ performs better than the macropacket scheme for independent channel losses when the loss rate is less than 20%. For larger loss rates the macropacket scheme seems to take the advantage, but for lower probabilities the data carousel effectively sends an equal number of packets into each of the parts without adverse extra overhead.

The viability of using divided RLF in a real application depends on the time complexity of the algorithm used to decode the RLF encoded packets in each of the parts. If a fast enough decoding algorithm (which would not work well for the whole data size) could be used, then, as the results indicate, the division into a couple of parts while in addition using the macropackets or the data carousel scheme would result in a low overall overhead.

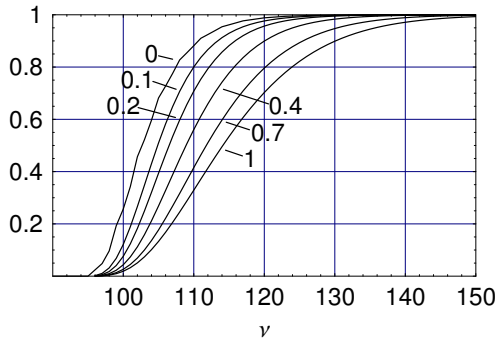


Figure 3.7: The decoding success probability using data carousel with $k = 3$ and $n = 32$ as a function of the number of received packets. Different channel loss probabilities are used to demonstrate the dependence of this scheme on the erasure probability p.

Table 3.6: Mean overheads for case $k = 3$ using macropackets and divided RLF.

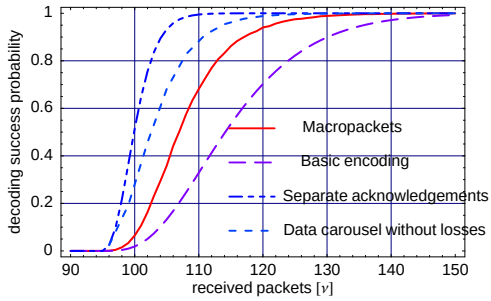
		Mean overhead		
n	$\rho(d)$	Separate acks	Macropackets	No macro
32	$\{0.89, 0.10, 0.01\}$	4.82 (5.0%)	12.5 (13.1%)	20.3 (21.1%)
64	$\{0.911, 0.08, 0.01\}$	4.82 (2.5%)	15.6 (8.1%)	26.0 (13.5%)
100	$\{0.936, 0.05, 0.014\}$	4.82 (1.6%)	18.2 (6.1%)	31.0 (10.3%)

3.4 A systematic code with belief updating

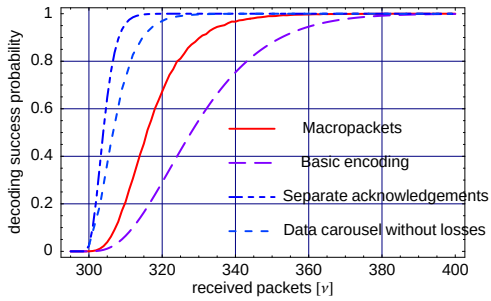
We will now move slightly away from the ideal fountain coding principle. In an ideal digital fountain, all of the packets are statistically equivalent, and a receiver can join anytime to start collecting the packets. When enough packets, $f \cdot k$ on average, are collected, the decoding of the original data can be completed.

In this section we abandon this time independence property and study how low an overhead is attainable by sequential repair packets, when an estimate of the channel erasure probability p is available. The coding is started by sending the original file blocks as such once, this round forms the systematic part of the code. The idea is to optimise the sequence of the repair packet degrees sent after the systematic part, when the sender continuously updates his belief about the status of the receiver. The method we study is based on a greedy decision making in the sense that each of the repair packets is optimised to advance the decoding process as much as possible immediately upon arrival. This strategy is studied in Publication 4.

After the systematic part, repair packets are sent until the decoding is complete. Following the spirit of LT coding, the repair packets are random linear combinations of d original blocks sampled over the whole message, where d is the repair packet degree. We retain the rateless property of the coding, as a potentially unlimited sequence of repair packets can be sent.



(a) $n = 32$



(b) $n = 100$

Figure 3.8: Basic encoding (plain divided RLF) compared to the use of macropackets, data carousel and separate acknowledgements in the case $k = 3$.

However, the repair packets are no longer statistically identical because the construct of the repair packets changes as the sender's belief about the state of the receiver evolves when more packets are sent.

The estimate of the channel erasure probability p is used to keep track of the state of the receiver. The sender calculates the probability of the receiver to have any particular number of decoded blocks and uses this information to optimise the degree of the repair packet. After the systematic part, repeating this optimisation procedure results in an optimal degree sequence for given p and file size k.

We can describe two different greedy strategies for finding good degree sequences. We will find the packet degree which maximises the probability of the next packet to include only one block yet unknown to the receiver, and thus giving the receiver the possibility to decode a new block.

Alternatively, we could optimise the packet degree for giving the highest probability of advancing to the 'fully decoded'-state at the recipient end. The idea is similar as in the **Max.Pr.** optimisation goal presented in Section 3.2. This kind of goal would have to take into account all of the packets the receiver has collected so far in order to optimise the packet degree, whereas the first goal leads to considering only such packets which have directly helped the decoder instantly decode a new block.

The first described goal is more convenient for our scenario. First of all, if there is more than one missing block at the receiver and no unprocessed packets in the buffer, it is impossible to advance to a decoded state with just a single extra received packet. The book-keeping we would have to do in the algorithm would additionally cause serious complexity issues, as we would have to somehow track a vast number of different states and calculate the state transition probabilities from these states to the decoded state. It is much easier to consider packets which are optimised just for including enough information to help the decoder to decode a new block, as we will see in Section 3.4.2.

3.4.1 Related research

The greedy encoding strategy is partly inspired by [KFMR05], which is a preliminary work to [KMFR06], in which the authors present Growth codes to increase data persistence in sensor networks. Growth codes are further considered in [DWR07], in which a method for unequal error protection (i.e., data in which some of the packets are more important than others) is presented with a video streaming application.

An alternative scenario to ours would be one where the sender knows the exact number of blocks the receiver has decoded. This approach is studied in [Con01], where the author presents an oracle who knows the exact number of already decoded blocks. This information is used to improve existing degree distribution by suggesting possibly better packet degrees for some of the generated packets. Thus, this work presents a heuristic for packet degree optimisation. The oracle-based packet degrees are chosen using the same criterion as in our scenario, i.e., by choosing the packet degree giving highest probability to reduce the degree to one immediately by decoding.

Further, in [BCMR04] the same greedy criterion is used in an approach to data reconciliation between downloaders in a peer-to-peer-type environment. The downloaders are allowed to *recode* already acquired packets and distribute these to peers possibly speeding up each other's downloads. The degrees of recoded packets are chosen between the greedy optimum and some upper limit to mitigate the possible inefficiency of choosing a local optimum.

Similar results as in Publication 4 are also presented in [BDS07]. The used optimisation criterion is similar as used in our work, and the publication time of [BDS07] coincided with that of finalising Publication 4.

3.4.2 The greedy method based on belief about the receiver's state

The greedy encoder differs from the LT encoder presented in Section 2.4 in how the degree of the packet is determined. The coding starts by sending the k original blocks as-is as systematic packets, then repair packets are sent with given degrees. A degree distribution is not sampled, but instead a packet degree sequence is calculated and is deterministic. The used packet degree depends on the channel loss probability p through the sender's belief about the state of the receiver. Next we study in more detail how to optimise the degree sequence of the repair packets.

Let us denote by $P(i,n)$ the probability of a randomly generated degree-i packet being such that it includes exactly one block the receiver has not yet decoded, given that there are n missing blocks in total. Further, let $\mathcal{P}_t(i)$ denote the unconditional probability of a degree-i packet to give the decoder at least one new, yet undecoded block at time t. The time is measured in discrete steps where $t = 0$ corresponds to the time instant when the initial systematic part is finished, i.e., all the k blocks have been sent as such. After this, each sent packet advances the time count by one.

The repair packets are optimised to reveal exactly one new, yet undecoded packet upon arrival at the receiver's side with the maximum probability. The exact optimisation criterion is

$$i^* = \arg\max_i \mathcal{P}_t(i) \ , \tag{3.27}$$

which we call the *greedy optimisation criterion*. Note that although the definition states the optimisation is done to reveal exactly one novel block, in practice the situation might be such that more than one block is decoded if an iterative decoder is used.

We can calculate $P(i,n)$ using a combinatorial argument. Exactly one of the i blocks must be chosen from the n missing ones and the rest $i-1$ blocks from the $k-n$ blocks the receiver has already decoded. Thus,

$$P(i,n) = \frac{n\binom{k-n}{i-1}}{\binom{k}{i}} \ , \tag{3.28}$$

and the unconditional probability can be written as

$$\mathcal{P}_t(i) = \sum_{n=0}^{k} P(i,n) f_{t-1}(n) \ . \tag{3.29}$$

The distribution $f_t(i)$ gives the probability of i missing blocks out of the total k at time t. Initially, at $t = 0$, after the systematic round, f_t is binomial:

$$f_0(n) = \binom{k}{n} p^n (1-p)^{k-n} \ . \tag{3.30}$$

The greedy optimisation criterion in Definition 3.27 is used to choose the repair packet degree at specific time t. The sender calculates the optimal degree, maximising (3.29), sends the corresponding packet and updates the belief (i.e., distribution f_t) about the state of the receiver,

$$f_t(n) = \big(1 - (1-p)P(i^*,n)\big) f_{t-1}(n) \tag{3.31}$$

$$+ (1-p)P(i^*,n) f_{t-1}(n+1) \ , \tag{3.32}$$

where we have taken into account the possibility of a successful repair packet to reduce the number of missing blocks and the possibilities of a non-useful repair packet and packet erasure, i.e., the cases when a new block cannot be decoded.

By using (3.27)-(3.32) we can calculate as long a degree sequence as needed. For a specific k and p the sequence is always the same an can be precomputed. However, if the estimate of p changes during the transfer, new degrees can be calculated on the fly.

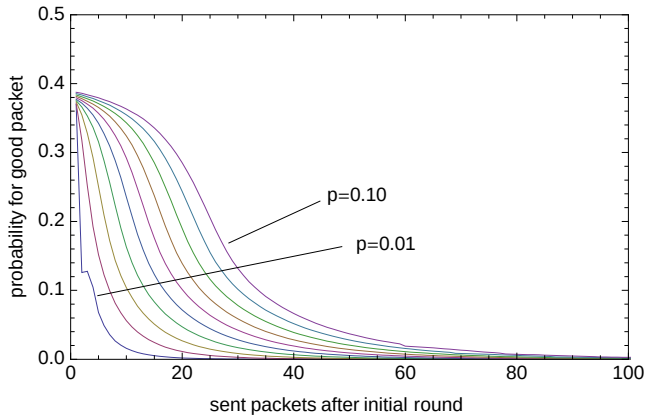


Figure 3.9: The probability of repair packet to reveal a new block for different p and number of sent repair packets after the initial systematic round for $k = 100$.

3.4.3 Decoding algorithm

The decoding is performed exactly as in LT decoding (cf. Section 2.4). In the derivation of the optimisation steps, we have not taken into account the buffered packets which do not immediately yield new decoded blocks.

3.4.4 Numerical results

As presented in Publication 4, the probability of the first repair packet after the initial round being helpful is roughly $1/e \approx 0.368$. This probability then decreases as more repair packets are sent at a speed that depends on the erasure probability p. This is depicted in Figure 3.9 for $k = 100$. Although the probability does not seem high, i.e., it is much less than 1, this greedy strategy yields good results in practice. When the packet losses are not too high, only a few repair packets need to be sent and the required overhead is low even for small data lengths of tens or hundreds of blocks. Figure 3.10 shows the probability of decoding failure in terms of overhead factor and Table 3.7 shows the simulation results for mean number of packets sent and received for decoding with different erasure probabilities.

As the size of the source data, k, increases, the required overheads again become lower and the decoder error performance better. For example, for $k = 100$ with 5% overhead the decoding does not succeed with probability 0.02, whereas for $k = 1000$ the same probability is only $7 \cdot 10^{-4}$. This demonstrates the asymptotic optimality of this scheme.

The nature of the degree sequences the greedy algorithm yields is depicted in Figure 3.11, in which the resulting sequences for $k = 100$ and different values of p are plotted. The average degree of the sequences is near 20 for all p. Typically, the degrees start at a low value near $1/p$, then grow a little and after a peak they decrease. Ultimately, for any p, the sequences form a saw-tooth like pattern, which always ends in sending a full, degree-k, packet. This kind of packet is guaranteed to decode the last missing block

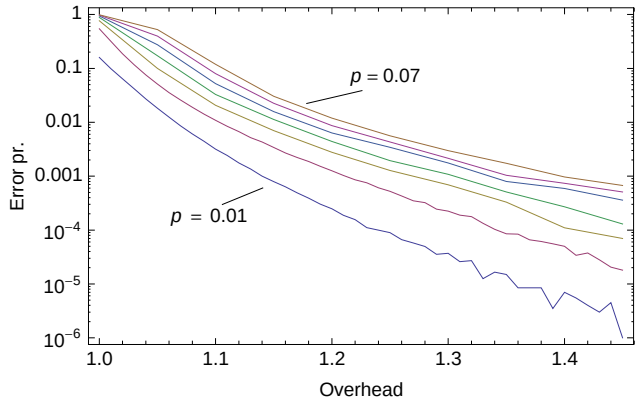


Figure 3.10: The decoder error probability for $k = 100$ and different p as the function of the overhead factor.

Table 3.7: Simulations for $k = 100$ with greedy sequences

	Received		Sent	
p	Mean	Std	Mean	Std
0.01	100.4	1.36	101.5	2.12
0.02	101.5	2.40	103.5	3.43
0.03	102.4	2.84	105.6	4.03
0.04	103.4	3.35	107.8	4.62
0.05	104.5	3.83	110.0	5.23
0.06	105.4	3.99	112.2	5.43
0.07	106.4	4.23	114.4	5.74
0.08	107.5	4.49	116.8	6.01
0.09	108.4	4.67	119.1	6.19
0.10	109.4	4.85	121.5	6.33

when all of the other blocks have been decoded, and thus it makes sense to send it every now and then.

These results demonstrate that the greedy erasure correction strategy is a viable erasure correction method for BECs. In Publication 4 we further discuss how the scheme performs when the estimated p differs from true channel conditions and briefly mention that the performance is decent even with burst errors. We also present a simplified algorithm with lower time complexity to calculate the degree sequences, with somewhat poorer performance in terms of required overhead. The simplification essentially keeps simpler belief about the state of the receiver at the sender's side, by truncating the used distribution f or even using only maximum likelihood estimation and setting $f_0(n) = \delta(p \cdot k)$, where δ is Dirac's delta function. This resembles the information the oracle gives in [Con01], but instead of exact knowledge we have only an estimate of the number of missing blocks.

Overall the results warrant similar kind of conclusions as are presented in [Con01]. By selecting a degree which maximises the probability for de-

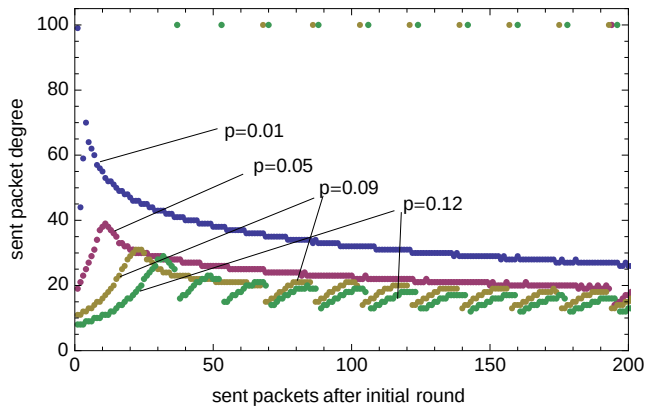


Figure 3.11: The greedy sequences for $k = 100$ and different p.

coding one new block, many of the resulting packets are redundant. However, in our scenario a systematic sequence is sent initially, and thus, with reasonably low channel loss probabilities, the inefficiency due to redundant packets is small when considering the total required overhead for the decoding process. Also, without full knowledge of which particular blocks are missing some redundancy cannot be avoided. Nevertheless, the greedy optimisation criterion does not necessarily result in globally optimal degree sequences.

3.5 Summary

In this chapter we have studied different settings in which packet erasure correction coding could be applied. We have studied different coding strategies and have analysed their performance and suggested some optimisation techniques to get the best possible performance with discussion of the advantages and possible disadvantages of the different schemes. In particular, we considered the optimisation of LT codes, random linear fountain and a sequential erasure correction strategy.

We began by considering the optimal degree distribution problem of LT codes with a very small message length. Exact analysis using Markov chains and a combinatorial approach was presented with the possibility of optimising for two different goals, either maximising the decoding probability with exactly k packets or minimising the expected number of overhead packets. Although in [BC08] the authors note that with robust soliton distribution and properly optimised parameters a similar performance can be achieved for small message lengths as with the methods presented in Publication 1 (also in Publication 2), we have presented and solved the degree distribution problem exactly for small message lengths without assuming anything about the form of the degree distribution. These results are novel and have not been presented in the literature before. We also noted that the two optimisation criteria result in slightly different degree distributions but a similar performance.

The degree distribution optimisation was further studied using an algorithm based on simulations, borrowing some ideas from importance sampling theory. This allowed us to get from the toy cases of very small k into message sizes of the order of hundreds of blocks. This method works by iteratively optimising the degree distribution used by calculating the gradient of an objective function and taking a step towards a better distribution. The general idea of using simulation results obtained with a given degree distribution to estimate the performance when a different degree distribution is used is intriguing and can potentially be used in other applications as well. However, in order for this optimisation strategy to work well, we have to make educated guesses about the form of the degree distribution. The results indicate that the form resembling robust soliton distributions works best out of all the different types tested. For message lengths of hundreds of blocks we found out that our suggested degree distribution works as well as, or slightly better than, robust soliton distribution.

These studies on the degree distributions of LT codes reveal the fact that only a few parameters are needed to define a well-performing degree distribution. Unfortunately, the methods we have used suffer from state space explosion or have time requirements too high for them to be used for longer message sizes. However, the goal was to study the small lengths, for which previously published degree distributions do not work well.

The rest of this chapter considered other kinds of erasure correction strategies based on the fountain principle. We presented a divided random linear fountain, where not acknowledging separate decoded parts causes the performance to deteriorate. We proposed and analysed two ideas for improving the performance, namely the use of macropackets, i.e., LT coding applied on top of divided RLF, and a data carousel-inspired strategy. As the results suggest, such strategies do lower the required overhead. The practicality of divided RLF depends on the availability of an efficient decoding algorithm for RLF in each of the parts the original data are divided into.

Finally, we took a step away from the ideal fountain coding principle and abandoned the requirements of the time-independence and statistical equivalence of the sent packets. We developed an erasure correction strategy in which each of the original blocks is first sent as such once. After the initial round, repair packets are sent to correct the possible gaps left in the sequence of the original blocks. As the code is systematic and the optimised repair packet degrees depend on the previously sent packets, the sent packets are not statistically identical.

The repair packet degree sequence was optimised by assuming that an estimate of the channel loss probability p is available. On the Internet, some protocols, such as the Datagram congestion control protocol (DCCP) [KHF06], can provide such estimations. The greedy criterion of optimising the packet degree for revealing exactly one new block at the receiver's end upon arrival works well in the simulation using a binary erasure channel. Complex degree distributions are not required; instead, a deterministic degree sequence is calculated, and typically the required overhead with the optimised sequences is $O(p)$. Although this method is not a true fountain code in the strict sense, as it is channel-dependent, an implemen-

tation could use an adaptive scheme in which the optimisation takes into account varying channel conditions, resulting in a robust error correction performance.

In conclusion, we have presented novel schemes for optimising existing fountain coding methods in different settings and presented a couple of our own ideas for new coding strategies. These methods can be used to improve and optimise the performance of fountain coding and related methods, resulting in as low an overhead as possible.

4 ERASURE CODING FOR REAL-TIME SCENARIOS

In this chapter we present and analyse packet erasure correction methods for streaming applications with real-time requirements. A sliding window is used for marking the section of the data (i.e., the source blocks) which contains the non-expired information. The proposed coding methods are systematic, each original data block is initially sent as-is, and repair packets are later sent to possibly fill in the gaps left by the packet erasures. The methods also follow the ideas of fountain coding. The encoding uses the same procedure as in LT coding and the decoding is done iteratively. We aim at code simplicity and good performance of the codes in terms of low an overhead and low encoding and decoding complexities. An example of an application which could benefit from these kind of codes is multimedia transmission, or, more specifically, different kinds of video and audio streaming services.

One drawback of the existing digital fountain coding methods is that the optimum performance is achieved only in the asymptotic case. For smaller source data sizes the overhead can be too large in practice, especially if the degree distribution is not carefully designed for the prevailing conditions of source size and the application requirements. Further, as traditional fountain codes send the packets as descriptions of all of the source data, no guarantees on the relevance of the received data at any particular time instant can be given to the receiver. Although versions of Raptor codes [Sho06] have been specified for small lengths for both file delivery and streaming applications, we study methods which are potentially more efficient.

Our methods employ a sliding window of an appropriate size for bounding a section of the source data to provide rough ordering and, especially, mark the part which the receiver can play out from the buffer used for receiving and decoding the data. The window size, w, is thus dictated by the application employing the scheme. In the optimisation methods presented in this chapter, we typically use example window sizes of tens of blocks, which is a realistic assumption for the size of the buffer for low-latency applications. For example, in a typical voice-over-IP application the payload size corresponds to 20 ms of voice. The number of outstanding packets of such a size should not be very large in order to keep the round-trip delay of a voice conversation sufficiently small.

The research problem we address is the generation of efficient repair packets. The codes send original data as-is once, i.e. they are systematic, and the repair packets are chosen to give immediate benefit in terms of novel decoded blocks upon arrival. As in the systematic code with greedy decisions presented in Section 3.4, we assume the sender has some estimate of the channel erasure probability p. The optimisation is performed for this specific erasure probability. However, the results indicate that the methods are quite insensitive to varying channel conditions; the specific code parameters optimised for a certain estimate of the channel loss probability, $\widehat{p}$, give good results even if the true p is different.

A major conceptual change in this chapter is the fact that with real-time applications one does not require the full decoding of all of the source data. In lossy channels some of the source data are inevitably lost, and streaming applications are designed to be robust against a small number of missing data blocks. Our aim is to achieve the best possible residual erasure performance, given the code properties and channel conditions. This performance criterion is different from that considered in Chapter 3. The methods that are presented are not true fountain codes as they have a defined rate, which has to be decided in advance. There is a clear trade-off between the design rate of the code and the erasure correction performance: the lower the code rate, the lower the residual erasure probability is.

The channel model used is again the binary (or packet) erasure channel, BEC, without feedback. However, packet losses are often correlated in real networks, a scenario not captured by the simple BEC model. Therefore, we will also present some simulation results with the proposed codes using the *Gilbert-Elliot model*, which is a two-state model in which both states have their own packet erasure probability. The methods we have developed are currently not able to optimise packets for the GE model, but the simulation results we present give us an insight into how the proposed methods would work in the presence of correlated packet erasures.

We start by presenting our contribution in more detail in Section 4.1. Then we will introduce the general procedure to use the sliding window in erasure coding in Section 4.3. We also define the encoding and decoding procedures involved. In Publication 5 we present two approaches for finding good fountain code-like erasure correction methods by fixed-point iteration. The repair packets are generated either by choosing degrees in half-windows, or then by sampling each of the window locations with independent probability. We cover these in Section 4.4. Publications 6 and 7 present two approaches based on first sampling a packet degree and then this number of blocks in the current window to be included in the repair packet, which is sent probabilistically after each systematic block. We analyse and optimise the performance of the coding using a Markov chain model for the state evolution of the coding process. The difference between these two methods lies in the encoding procedure. One of them uses uniform sampling within the window and the other one is based on the biased sampling of the source blocks. These methods are presented in Section 4.5.

4.1 Contribution

We study the use of a sliding window in different scenarios in which we aim at as low as possible a residual erasure probability Q_r, and the trade-off between the code rate R and Q_r is the research question we address. The code rate is controlled either by a deterministic scheme, in which a repair packet is sent after every s sent blocks, or by a probabilistic strategy, in which a repair packet is generated and sent with a probability P after each systematic block. For the purpose of the analyses, we assume that the stream is very long in order to be able to assume that the system is in a steady-state situation.

The first scheme that is presented uses a sliding window with half-

window step sizes, in which a repair packet is deterministically sent after each window movement. The repair packets are generated by separately sampling both of the half-windows, optimising the number of blocks (d_1, d_2) picked from each of the halves, with the total packet degree $d = d_1 + d_2$. This is a more rigid strategy with not as much of a probabilistic flavour compared to the scenarios presented later in this chapter, but the exact analytical calculations can be performed using the independence of the number of missing blocks in each of the half-windows.

After this we move to a strategy in which repair packets are sent with a probability P after each window movement of one step. Instead of the blocks being picked from half-windows, the whole current window is considered when the stream blocks to be included in a repair packet are sampled. Every position of the window has its own probability of being sampled and included in the repair packet. These probabilities are optimised to give the best possible performance. The repair packet sending probability P controls the code rate.

These two methods are presented and analysed in Publication 5. The sender keeps track of his belief about the state of the receiver, and the cycle of optimisation (in the greedy sense), belief update and repair packet sending is repeated. With the help of fixed-point iterations, stationary solutions are found and can be used for the degree optimisations. This work can be seen as a continuation of Publication 4 with similar ideas about the bookkeeping of the receiver's state and packet degree optimisation in a greedy fashion.

The last part of the thesis consists of an analysis of the sliding window erasure correction, in which the general methodology is to send a repair packet probabilistically after each sent systematic packet. In contrast to the previous method, the chosen degree is first sampled from a degree distribution, as the results in Publication 5 suggest that the degree-based approach works better than giving each of the window positions an own probability to be sampled into a repair packet. The performance with given degree distributions is analysed using a Markov model. This model is derived by considering the steps of window movement, systematic sending, and repair packet generation. For each of these steps we derive state transition probabilities, in which the state is the pattern of blocks the receiver is lacking. We keep the window size small and limit the allowable number of missing blocks to three in order to avoid a state space explosion. This limitation results in some truncation error in the analytical results, but this has been proved to be negligible for the scenarios we are studying by comparing the analytical results to the simulations.

Two different scenarios are studied extensively using a Markov model. First, we present a method using the LT encoding algorithm with uniform sampling inside a window, studied in detail in Publication 6. We extend this scheme by allowing biased sampling of the blocks inside the window, still sampling the degree d in advance from a given degree distribution. The decoding probability achieved using biased sampling is calculated with the help of Wallenius' hypergeometric distribution [Wal63]. This development is presented in Publication 7.

One of the most notable findings from our numerical studies is that

a single-degree distribution works best in those kinds of situations with a sliding window approach. Even more, when biased sampling is allowed, a fixed, deterministic pattern of window positions to be picked for inclusion into the repair packets seems to be the optimal way to perform the coding. Thus, the degree distribution problem which makes the analysis and optimisation of LT and Raptor codes somewhat complicated, can be avoided in the scenarios we present.

The performance of the presented schemes is shown to be on a par with, or even better than, Raptor coding. However, the assumption that a good estimate of the channel erasure probability is available for optimisation purposes is a clear deviation from the ideal fountain coding principle and hinders the suitability of the codes for multicasting over multiple channels with different properties. Additionally, as the presented sliding window-based methods are systematic, not all of the packets that are sent are stochastically identical descriptions of the source data, another violation of the ideal fountain principle.

4.2 Related research

Adaptation of LT coding for real-time traffic using a sliding window algorithm is dealt with in [BCG+07]. The authors present an encoding scheme called sliding fountain, in which a sliding window partitions the source data and marks the section from which the LT encoder picks the data blocks. This idea is similar to the sliding window encoding we present in Section 4.3.

The Raptor codes [Sho06] used in specifications, such as the recommendation for erasure coding in multimedia broadcast/multicast service (MBMS) in 3GPP [3GP07] and in RFC 5053 [LSWS07], are defined for small message lengths, from $k = 4$ up to $k = 8192$. These codes work reasonably well in the defined range and are efficient in terms of the overhead and also computational complexity. However, especially the overhead can be made lower using the optimisation methods and coding schemes presented in this chapter.

FEC coding for streaming video has been studied in many different publications. General FEC performance studies are presented in [Fro01]. An overview of video streaming codes and strategies over the Internet is given in [WHZ+01], in which the fountain codes are mentioned to be an efficient strategy because of the independence from the loss patterns. Scalable video streaming using fountain codes is discussed in [WCF06] and the use of fountain codes in media streams in [JS05].

4.3 Sliding window algorithm

All of the erasure codes considered in this chapter use a fixed size sliding window algorithm for guaranteeing the real-time requirements imposed by the used application. The window size w is measured in source blocks. Contrary to what is assumed for example in [BCG+07], w is not a free parameter but dictated directly by the requirements of the application.

Let us denote by $\{\ldots, b_{i-2}, b_{i-1}, b_i, b_{i+1}, b_{i+2}, \ldots\}$ the stream of the

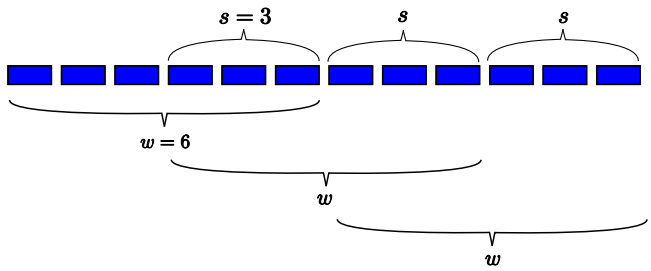


Figure 4.1: The sliding window depicted. Step size s determines how many blocks the window advances after the current content of the window is processed.

source blocks, and the current sliding window by $\{w_1, w_2, \ldots, w_w\}$. Essentially w_i is a pointer to the stream block currently in position i of the sliding window. Note that the stream is considered to be practically infinite from the analysis point of view. The sliding window operates as follows:

1. Move the window s steps forward by introducing s new subsequent blocks into the window and drop the last s blocks out, i.e., set $\{w_1 = b_i, w_2 = b_{i+1}, \ldots, w_s = b_{i+s}\}$.
2. Send systematic packets $w_1, w_2, \ldots, w_s$.
3. Process the window as defined by the used coding scheme (i.e. possibly generate a repair packet and send it).
4. Go to 1.

Figure 4.1 illustrates the sliding window algorithm.

4.3.1 Encoding

As the codes are systematic, immediately after the sliding window moves, s stream blocks b_i are sent intact as systematic packets. After the systematic packets, a repair packet is generated with probability P. Thus, the parameters s and P define the effective code rate:

$$R = \frac{s}{s+P} \quad , \tag{4.1}$$

as for every s sent systematic packets, on average P repair packets are sent. Figure 4.2 depicts the relation between P and R when $s = 1$ and between s and R when $P = 1$. These are the two scenarios we will consider in this chapter.

The encoding procedure of a repair packet is generally the same one as is used in the LT encoding, presented in Section 2.4. A degree distribution ρ is sampled and d blocks are added together using bitwise XOR:

$$r = \bigoplus_{j=1}^{d} w_{i_j} \quad . \tag{4.2}$$

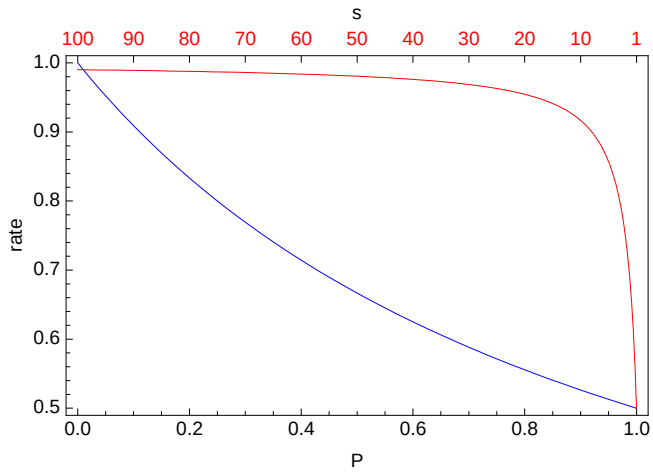


Figure 4.2: The resulting effective code rate with $s = 1$ (lower line) and variable P and with $P = 1$ and variable s (higher line).

The average degree of all sent packets is

$$d_{\text{avg}} = \left(s + P \cdot \sum_{i=1}^{w} i\rho_i\right)/(s + P) \ . \tag{4.3}$$

In contrast to traditional LT encoding and other proposed erasure coding methods in the literature, we will also study scenarios in which we allow biased sampling of the source blocks, as opposed to the uniform sampling employed in LT codes. However, this does not modify the principle of first sampling a packet degree and then choosing the blocks to be included.

An exception to the degree-based sampling is the second scenario studied using fixed-point iteration in Section 4.4, in which repair packets are actually formed by including each of the window positions independently of the others with potentially different probabilities p_i. In a way this resembles the encoding algorithm of the random linear fountain in which, however, each of the blocks is considered to be included with equal probability $1/2$.

4.3.2 Decoding

The decoding is performed iteratively as in LT coding. The information on which particular blocks are included in a repair packet is conveyed to the receiver, and the decoding of the original source blocks can be depicted using the graph based approach. As discussed in Chapter 2, the decoding can be performed on different levels of completeness. We will consider the following decoding algorithms:

1. Decoding using only those packets which immediately reveal new symbols (cf. the greedy criterion in 3.4).

2. Store those repair packets which are not immediately useful in a buffer. Purge all those packets from the buffer which include expired (i.e., out-of-window) blocks.

3. Full iterative decoding in which no buffered repair packets are discarded. Allows decoding using out-of-window blocks.

These methods are suboptimal as full solving of a linear system is not considered. The repair packets are generally chosen to reveal new decoded block immediately with high probability, and thus it is unlikely the sliding window methods would benefit from such a complete decoding scheme. It should be noted that the simplest decoding strategy 1 is the one explicitly assumed in the performance analyses. The latter two algorithms are used in some of the simulations to provide comparison and study the possible improvement in the performance provided by the more complete decoding strategies.

4.3.3 Optimisation and repair packet generation

In the repair packet generation, we have chosen to use similar greedy criterion as was presented in Section 3.4. However, now the goal of the optimisation is to attain as low a residual erasure probability Q_r as possible. All the repair packets are optimised in a way to give immediate benefit for the receiver in terms of novel decoded blocks with the highest probability possible. This means that the buffering of packets is not considered in the following analyses.

In general terms, a correction packet which is immediately useful includes exactly one yet undecoded block while the rest of the blocks are known and decoded by the receiver. If the receiver has k missing blocks in the sliding window, i.e., $w-k$ blocks have arrived and are decoded, then in the uniform sampling case the probability that i blocks, where $0 \leq i \leq w-k$, which are yet undecoded are picked to be included in a repair packet of degree d, is

$$P_i(d,k) = \frac{\binom{k}{i}\binom{w-k}{d-i}}{\binom{w}{d}} \quad . \tag{4.4}$$

The explanation is combinatorial: from k missing blocks i are first chosen, followed by choosing the rest $d-i$ blocks from the $w-k$ already received ones, giving total number of ways to generate packet with i yet undecoded blocks. Finally this result is divided by the number of all possible packets.

As already noted, the condition for a repair packet to be immediately useful is that it includes exactly one block yet unknown to the recipient; all the other chosen blocks should be already decoded. Thus, in general, repair packet degrees which maximise the probability

$$P_1(d,k) = \frac{k\binom{w-k}{d-1}}{\binom{w}{d}} \tag{4.5}$$

are used.

The calculation of biased sampling probabilities will be presented in Section 4.5.2 later in this chapter.

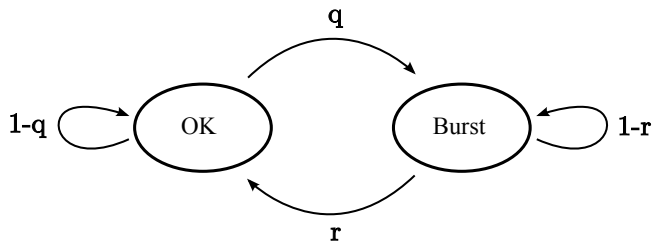


Figure 4.3: The Gilbert-Elliott error model. The system can be in two states, both of which have their own erasure probability.

4.3.4 Error models

Packet losses are often correlated in real-life scenarios. For example, if the losses occur because of network congestion, there are often multiple drops in a row, or a period when drops are much more likely than under normal conditions. For wireless and wired links the bit errors are often modelled using *Gilbert-Elliott (GE) model* [Gil60, Ell63]. We model the bursty losses on packet level using the GE model.

Although the analysis and repair packet optimisation is performed assuming independent packet losses (the packet erasure channel model), we will perform some simulations using the GE model in order to evaluate the performance of the proposed methods under different network conditions.

In the GE model, the channel is either in a bad or in a good state, also called *OK* and *Burst* state, both of which have their own packet loss probability, p_a and p_b, respectively. The losses in both of the states are assumed to be independent. The state transitions are considered after each sent packet, regardless of if it is dropped or not. We use the model with independent losses for each state, with packet erasure probabilities p_a and p_b in good and bad state, respectively. The changes of channel state are independent of previous states, and thus the model constitutes a two-state discrete time Markov chain, depicted in Figure 4.3. The state transition matrix is

$$\boldsymbol{P} = \begin{pmatrix} 1-q & q \\ r & 1-r \end{pmatrix} , \tag{4.6}$$

and the corresponding stationary state probabilities $\boldsymbol{\pi} = (\pi_{\text{ok}}, \pi_{\text{burst}})$ are

$$\begin{cases} \pi_{\text{ok}} = \frac{r}{q+r} \\ \pi_{\text{burst}} = \frac{q}{q+r} \end{cases} . \tag{4.7}$$

The average packet loss probability then is

$$p_{\text{avg}} = p_a \cdot \pi_{\text{ok}} + p_b \cdot \pi_{\text{burst}} . \tag{4.8}$$

This quantity will be used in comparison to results obtained with independent packet drops where $p = p_{\text{avg}}$.

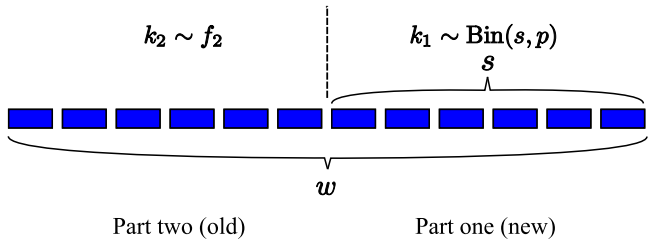


Figure 4.4: Illustration of the situation with $s = w/2$. The new part includes the blocks recently introduced into the window. k_1 and k_2 denote the number of missing blocks in the respective half-windows.

4.4 Half-window and probabilistic repair packet generation

In this section we will consider two separate scenarios and their analysis using fixed-point iteration, presented in Publication 5. We begin by considering the case with step size $s = w/2$ and $P = 1$, i.e., the repair packets are sent deterministically after every $w/2$ sent systematic packets. For this scenario we can write fixed-point equation taking benefit of the independence of the number of packets missing from each of the half-windows.

In the second scenario assume continuous window movement with $s = 1$ and allow probabilistic sending of the repair packets after each sent original block, i.e., P can be chosen freely. Further, the repair packets are generated by sampling each location i of the current window independent of the others with probability p_i, in which the probabilities for different locations are allowed to be different.

4.4.1 Half-window step size

Figure 4.4 illustrates the situation for the first scenario to be analysed. The window is divided into two different parts, 1 and 2. The new blocks after the window movement are found in part 1. Part 2 consists of the blocks which already have had one repair possibility (i.e., a sent repair packet) after the initial introduction to the window. After the window movement, the number of missing blocks k_1 in the half-window which includes the newly introduced blocks is binomially distributed because of independent losses in the channel. The distribution is denoted f_1. The number of missing blocks k_2 in the older part of the window is independent of k_1 but its distribution, f_2, is yet unknown.

The repair packets in this scenario are generated by sampling d_1 blocks from part 1 and d_2 blocks from part 2 and combining all of these sampled blocks into a single repair packet. Both of the parts are sampled uniformly at random. The strategy is to choose

$$(d_1^*, d_2^*) = \underset{(d_1,d_2)}{\arg\max}\, Q(d_1, d_2) \ , \tag{4.9}$$

where $Q(d_1, d_2)$ is the probability of the correction packet to include exactly one yet undecoded packet from one of the half-windows. This kind of

packet would result in immediate decoding in either of the halves. See Publication 5 for full form of $Q(d_1, d_2)$.

The probability distribution f_2 of the number of missing packets k_2 in the second part is still unknown, and a fixed-point iteration is used to find it in a stationary state. Note that f_1, the binomial distribution, is transformed into f_2 when an unknown block in the repair packet is from the half-window with newer blocks. To this end, we calculate

$$\begin{aligned} U(k_1) &= (1-p)Q(d_1^*, d_2^*) \sum_{k_2=0}^{n} f_2(k_2) \, \frac{P_0(d_2^*, k_2) P_1(d_1^*, k_1)}{Q(d_1^*, d_2^*)} \\ &= (1-p) \sum_{k_2=0}^{n} \left[f_2(k_2) P_0(d_2^*, k_2) \right] \cdot P_1(d_1^*, k_1) \ , \end{aligned} \tag{4.10}$$

for the conditional probability that a repair packet is useful and the contained unknown block is specifically from the newer half-window. P_0 and P_1 are given by (4.4).

The iteration itself is then defined by the distribution function update rule

$$f_2(k_1) = U(k_1) \cdot f_1(k_1 + 1) + (1 - U(k_1)) \cdot f_1(k_1) \ , \tag{4.11}$$

where the probabilities are updated on every window movement. Since $U(k_1)$ is a function of f_2, this is a relation of the form

$$f_2 = F(f_2) \ , \tag{4.12}$$

i.e., a fixed-point equation, which can be solved using iteration. On window movement, the blocks in part 2 fall out of the used window. Thus the distribution f_2 can be used to calculate the residual erasure probability of blocks inside part 2. When f_2 is solved using the iteration, the optimal degrees can be calculated, defining a proper erasure correction strategy.

4.4.2 Probabilistic repair packets

In addition to the half-window case, Publication 5 presents a method in which each of the window positions have independent probability p_i to be picked into a repair packet. We further have $s = 1$ and keep P as the defining parameter of the code rate (4.1). This can also be seen as a refinement of the half-window method, by introducing w different categories. These categories just happen to be of size one, and the degree decision simplifies to using a single probability p_i for each of the categories.

We use q_i to denote the probability of the block at window location i being still undecoded. For the block most recently introduced into the window we have $q_1 = p$, the channel erasure probability. The locations of the older blocks, $i > 1$, have a lower q_i, as they have had chances to become recovered. We then optimise the repair packets, i.e., the probabilities p_i, so that the generated repair packet is immediately useful for the decoder with maximum probability.

A particular block is undecoded and chosen for a repair packet with probability $p_i q_i$. Thus, with probability $1 - p_i q_i$ the block at location i is

either decoded or not chosen or both. We make an approximation that the events that two different blocks i ad j remain undecoded are independent. This is not the case in reality, as the probabilities that certain locations are decoded have mutual dependencies, and the assumption introduces some error in the analysis.

However, keeping with the approximation we can write the probability that a new block can be decoded upon repair packet arrival as

$$\begin{aligned} F = (1-p)\,((p_1q_1 \cdot (1-p_2q_2)\cdots(1-p_nq_n)+ \\ (1-p_1q_1)\cdot p_2q_2 \cdot (1-p_3q_3)\cdots(1-p_nq_n)+ \\ \vdots \\ (1-p_1q_1)(1-p_2q_2)\cdots p_nq_n)) \\ = (1-p)\cdot\sum_{i=1}^{n}\left(p_iq_i\prod_{j\neq i}(1-p_iq_i)\right) \quad . \end{aligned} \tag{4.13}$$

Thus we want to optimise

$$p_i^* = \arg\max_{p_i}\sum_{i=1}^{n}\left(p_iq_i\prod_{j\neq i}(1-p_iq_i)\right) \quad . \tag{4.14}$$

It can be shown that the solution for this is obtained by setting $p_1, p_2, \ldots$ equal to one as long as the condition

$$\sum_{i=1}^{k}\frac{q_i}{1-q_i} \geq 1 \tag{4.15}$$

is satisfied for a certain k. The rest of the p_i are set to 0, resulting in $p_1 = 1, p_2 = 1, \ldots, p_k = 1, p_{k+1} = 0, p_{k+2} = 0, \ldots$.

The update rules for q_i are

$$\begin{cases} q_1 \leftarrow p. \\ q_i \leftarrow \left(1-(1-p)\cdot P\cdot p_i\prod_{j\neq i}(1-p_jq_j)\right)q_{i-1} \quad . \end{cases} \tag{4.16}$$

This rule gives q_i as a function of q_{i-1}, so it can be used as a fixed-point equation to calculate values of q_i. Alongside with (4.16), (4.15) is used to give the optimal sampling probabilities.

Thus, the form of optimal sampling pattern always uses the first k blocks in the window in the repair packet and none of the rest. Basically, even though biased sampling is allowed, the method degenerates into forming deterministic repair packets.

4.4.3 Results and discussion

Figure 4.5 depicts the performance of the half-window method, in terms of the residual channel loss probability vs. the residual erasure probability Q_r. This can be compared to the performance obtained with the probabilistic

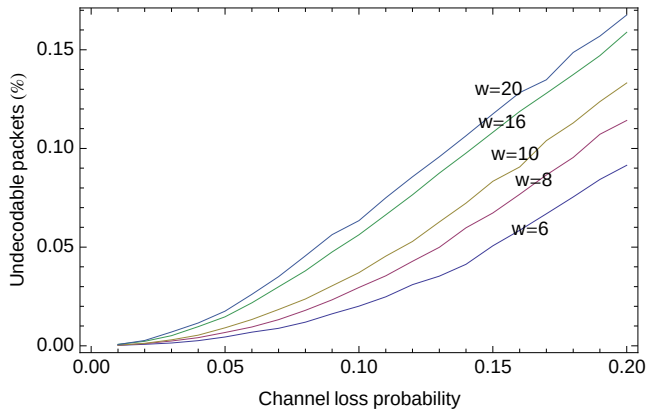


Figure 4.5: Results using the half-window method. A stream of 200000 packets was simulated using different values of p and different window sizes w.

method depicted in Figure 4.6. The half-window method generally gives better results, i.e., lower residual erasure probability than the probabilistic method in comparable situations.

For the half-window method, smaller window sizes result in better performance as the rate of the code is the lower the smaller the used window is. For example, for $w = 6$, the code rate is $R = 3/4 = 0.75$. From Figure 4.5, with $w = 6$, we see that for $p = 0.1$, we get residual erasure probability of 0.02 effectively meaning 80% fewer lost packets. The used degrees in both of the half-windows depend on the channel loss probability. The optimised degrees in part 1 start from approximately $w/2$ for low p and in part 2 from $w/4$. As the loss probability increases, the used degrees tend towards zero. This is natural because the higher the erasure probability, the more missing packets in part 1 (and 2) we have and the lower packet degree that is required to form a packet with exactly one missing block. The degrees in part 2 eventually converge to zero, resulting in a strategy in which the erasures are corrected only in part 1 of the window.

The probabilistic method gives a worse performance. The fact that the optimisation of the probabilities p_i relies on using the independence approximation in (4.13) is likely to be a part of the reason for this. As formulated in Publication 5, the optimum of (4.14) is achieved by always choosing some number of the newest blocks in the window while leaving the rest out of the repair packets.

As the half-window method with fixed degrees gives better results, we continue the studies of erasure coding for real-time streaming media by considering a scheme with probabilistic sending of repair packets and repair packet generation using degree distributions.

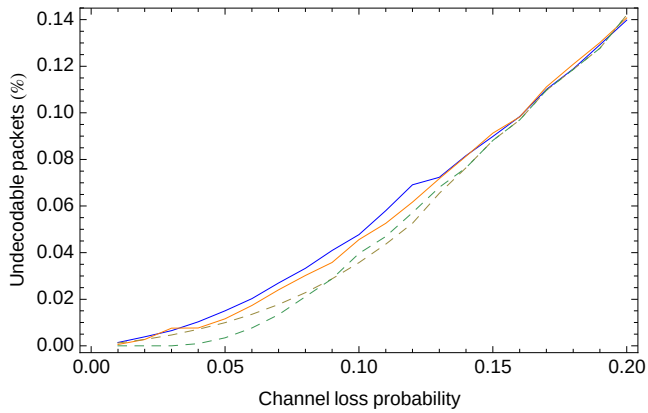


Figure 4.6: Results using the probabilistic repair packet strategy. The dotted lines correspond to theoretical results and solid lines to simulation results. Two different window sizes $w = 10$ and $w = 50$ were used, upper and lower lines, respectively.

Table 4.1: Number of states

	window size w			
k	5	10	15	20
w	32	1024	32768	1048576
3	26	176	576	1351
2	16	56	121	211

4.5 Markovian model for degree-based repair packets

In this section we focus on schemes in which the window advances one block at a time, i.e., $s = 1$, and the repair packets are sent probabilistically with probability P after each sent systematic packet. The sending probability P is the parameter defining the code rate. In contrast to the previous methods, we now keep track of the state of the receiver by considering the positions of the missing blocks in the current window. The two different repair packet generation methods considered are uniform and biased sampling of the stream blocks. These are studied in detail in Publications 6 and 7, respectively.

The window movement, systematic packet sending and repair packet sending and possible erasure correction cycle together can be perceived as a step in a Markov chain. Let us denote the state of this chain by the listing the indices of the packets the receiver is missing from the current window $\boldsymbol{i} = (i_1, i_2, \ldots, i_k)$, where k is the total number of currently undecoded packets.

Again we face the problem of state space explosion if the number of the

missing blocks in the current window is allowed to grow without a bound. The total number of states is 2^w. Therefore, for a tractable analysis we truncate the state space by limiting the number k of currently missing blocks from the window at any given time. Table 4.1 lists the number of possible states with different window sizes w and $k = 2$ or 3. Exact analysis using three missing blocks is still viable, and we will be using $k = 3$ when considering the numerical results and finding the optimal erasure correction strategies. This limitation will result in some errors in the analysis. However, when the size of the sliding window is reasonably small and the channel loss probability is low, the error is not significant as will be confirmed by simulations. Actually, the determining factor is the product of the window size w and p. If this product is less than k, it can be safely assumed that the analysis gives fairly accurate results.

Let us denote by $\boldsymbol{P}_1, \boldsymbol{P}_2$ and $\boldsymbol{P}_3$ the state transition matrices corresponding to window movement, systematic packet sending, and repair packet sending along with possible erasure correction, respectively. The state evolution is observed over the cycle of these three steps. In an infinitely long stream, the state probabilities $\boldsymbol{\pi}$ in equilibrium can be solved from

$$\boldsymbol{\pi} = \boldsymbol{\pi} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{P}_2 \cdot \boldsymbol{P}_3 \ , \tag{4.17}$$

together with the normalising condition $\boldsymbol{\pi} \cdot \boldsymbol{e}^{\mathrm{T}} = \mathbf{1}$, where $\boldsymbol{e}$ is the all-ones vector. The performance quantity of interest is the residual erasure probability Q_r, i.e., the probability for a certain block in the stream to remain undecoded when it drops out of the window. From the equilibrium state distribution we can calculate Q_r as

$$Q_r = \sum_{\boldsymbol{i}: i_k = w} \pi_{\boldsymbol{i}} \ , \tag{4.18}$$

i.e., by summing the probabilities of such states in which the block corresponding to the last position in the sliding window is missing from the receiver.

Next we consider the exact transition probabilities for the three different parts of the cycle and define the matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ and $\boldsymbol{P}_3$.

4.5.1 State transitions

Let us first consider the state transitions happening as a result of window movement. When no blocks from the current window are missing, we have only the self-transition

$$P_1\{\emptyset \to \emptyset\} = 1 \ . \tag{4.19}$$

For $k \geq 1$ we have

$$\begin{cases} P_1\{(i_1, \ldots, i_k) \to (i_1 + 1, \ldots, i_k + 1)\} = 1 & \text{if } i_k < w \\ P_1\{(i_1, \ldots, i_k) \to (i_1 + 1, \ldots, i_{k-1} + 1)\} = 1 & \text{if } i_k = w \ . \end{cases} \tag{4.20}$$

Systematic sending occurs after every window movement step. The

state transition probabilities are:

$$\begin{cases} P_2\{\boldsymbol{i} \to \boldsymbol{i}\} = 1 - p \\ P_2\{(i_1, \ldots, i_k) \to (1, i_1, \ldots, i_k)\} = p \quad \text{if } k < 3 \\ P_2\{\boldsymbol{i} \to \boldsymbol{i}\} = 1 \quad \text{if } k = 3 \ , \end{cases} \tag{4.21}$$

in which the last state transition probability is used to limit the number of possible errors and thus truncate the state space.

Lastly, the state transitions due to possible repair packet of degree d have the probabilities:

$$\begin{cases} P_3(d)\{\emptyset \to \emptyset\} = 1 \\ P_3(d)\{\boldsymbol{i} \to \boldsymbol{i} \backslash i_j\} = P(1-p)q(d, \boldsymbol{i}, i_j) \quad \text{if } j = 1, \ldots, k \\ P_3(d)\{\boldsymbol{i} \to \boldsymbol{i}\} = 1 - \sum_j Pq(d, \boldsymbol{i}, i_j)(1-p) \ , \end{cases} \tag{4.22}$$

where $\boldsymbol{i} \backslash i_j$ denotes a state with index i_j deleted from $\boldsymbol{i}$. $q(d, \boldsymbol{i}, i_j)$ is the probability of a degree-d repair packet to immediately reveal a missing block i_j (and this block only) in state $\boldsymbol{i}$. To obtain the full $\boldsymbol{P}_3$ we sum $\boldsymbol{P}_3(d)$ over all possible degrees, i.e., weighting with the corresponding degree probabilities,

$$\boldsymbol{P}_3 = \sum_{d=1}^{w} \rho(d) \cdot \boldsymbol{P}_3(d) \ . \tag{4.23}$$

All of the possible state transition are now accounted for, but we still need to consider the erasure correction probability $q(d, \boldsymbol{i}, i_j)$ to apply (4.17) and (4.18) and calculate the residual erasure probability Q_r with a given degree distribution.

4.5.2 Probability of erasure correction – uniform and biased sampling

In Section 4.3.3 we presented the optimisation criterion for picking efficient repair packet degrees as maximising the probability (4.5). When uniform sampling is used in the repair packet generation, the probability $q(d, \boldsymbol{i}, i_j)$ in (4.22) of degree-d repair packet to be of such form it can be immediately used upon receiving can be derived similarly. When the receiver is missing k blocks from the current window, and the index of the missing block to be repaired is i_j, we have

$$q(d, \boldsymbol{i}, i_j) = \frac{\binom{w-k}{d-1}}{\binom{w}{d}} \ , \tag{4.24}$$

where $k = |\boldsymbol{i}|$. This probability is the same as given in (4.5), without the multiplier k, as the yet undecoded block (of the k possible) is exactly the one at index i_j, so there is no choice to be made. The rest $d-1$ blocks are picked from the $w-k$ already decoded ones, and the total number of possible packets is $\binom{w}{d}$.

This result is valid for uniform sampling, but we are also interested in the scenario in which biased sampling of source blocks in the different window locations is allowed. For calculating the probability like (4.24) for this

case, a model for weighed sampling from hypergeometric distributions is used. Such a model is originally presented in [Wal63], and later extended to cover multivariate case in [Che]. We will refer to the distribution as Wallenius' noncentral hypergeometric distribution as suggested in [Fog08].

First the sampling weights used in the biased sampling need to be defined. The sampling weight distribution $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_w)$ defines the probabilities ω_i which are used by the encoder to sample d blocks inside the sliding window. Probability ω_i is the unconditional probability that the source block at window location i is sampled for inclusion in the repair packet. The weight distribution $\boldsymbol{\omega}$ could be considered to be, for example, the size distribution of different types of balls in a bin and the probability that a certain randomly chosen ball is picked is dependent on the ball size. Further, as the number of balls is finite, the remaining weight of the balls inside the bin will affect the subsequent samples.

Let the number of different categories, e.g., sizes, of balls be w and let the vector $\boldsymbol{m} = (m_1, \ldots, m_w)$ denote the number of balls in different categories as components. Then, the probability of a vector $\boldsymbol{x} = (x_1, \ldots, x_w)$ of number of sampled balls in each category is given by the Wallenius' distribution:

$$P(\boldsymbol{x}) = \left(\prod_{i=1}^{c} \binom{m_i}{x_i} \right) \int_0^1 \prod_{i=1}^{c} \left(1 - t^{\omega_i / D(\boldsymbol{x})} \right)^{x_i} \, dt \ , \tag{4.25}$$

where $D(\boldsymbol{x}) = \sum_{i=1}^{w} \omega_i (m_i - x_i)$ is the total weight of the remaining samples in the bin.

For the erasure coding problem using a sliding window, (4.25) can be simplified. The number of categories is w, the size of the window, giving for each of the window positions a unique probability to be sampled. However, the number of possible choices in each category is now one, the stream block corresponding to the window position. That is, $m_i = 1 \, \forall i$. Further, $\boldsymbol{x} \in \{0, 1\}^w$, as each of the window positions is either picked or not for a repair packet calculation. The simplified version of (4.25) then is

$$P(\boldsymbol{x}) = \int_0^1 \prod_{i : x_i = 1} \left(1 - t^{\omega_i / D(\boldsymbol{x})} \right) \, dt \ . \tag{4.26}$$

The final step is to consider how to calculate the erasure correction probability $q(d, \boldsymbol{i}, i_j)$. For a specific state transition, we need to sum (4.26) over all $\boldsymbol{x}$ which are of such a form that upon receiving of the corresponding repair packet the receiver can decode a block.

$$q(d, \boldsymbol{i}, i_j) = \sum_{x_{i_j} = 1, x_{i_l} = 0 \ \forall l \neq j} P(\boldsymbol{x}) \ . \tag{4.27}$$

If every position has a distinct sampling probability this is a computationally very demanding calculation as several evaluations of the integral (4.26) are needed.

The state transitions matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ and $\boldsymbol{P}_3$ are now fully defined state transition matrix for the Markov chain can be calculated. The equilibrium probabilities $\boldsymbol{\pi}$ can be solved using (4.17), and further used in (4.18) to get the residual erasure probability Q_r, given some degree distribution $\rho(d)$.

Table 4.2: Degree distributions used in studying the truncation error for $w = 15$.

Name	Distribution
Uniform	$\Omega_i = 1/15 \approx 0.067 \quad \forall i$
All-1	$\Omega_1 = 1$ and $\Omega_i = 0$ for $i = 2 \dots w$
Hi-low	$\Omega_1 = 0, \Omega_2 = 0.4, \Omega_{15} = 0.4$ and $\Omega_i = 1/12 \approx 0.0167$ for $i = 3 \dots 14$

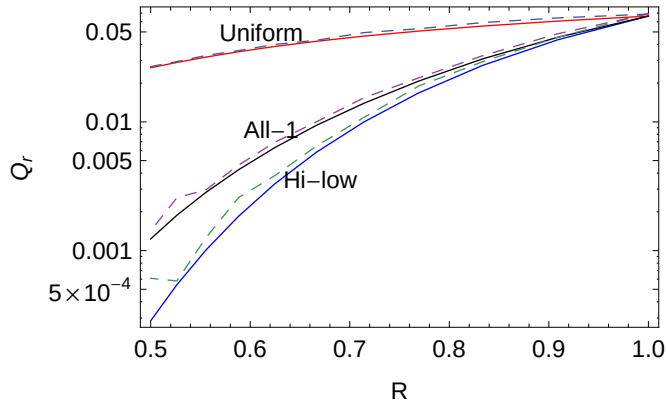


Figure 4.7: Study of the effect of the truncation error. Comparison of analytical and simulation results for $w = 15$ and $p = 0.15$.

4.5.3 Truncation error

First of all, we study the truncation error introduced by limiting the number of allowable errors to three in a window for $w = 15$ by comparing analytical results with those from simulations. Table 4.2 lists different types of degree distributions, which are used in the comparison. The performance measure Q_r is depicted in Figure 4.7 for $p = 0.15$ as a function of the code rate. We see that for the used parameters the simulation and the analytical results are close to each other. Similar results are achieved with other (w, p) pairs as long as $w \cdot p$ is low enough, of the order of three. For uniform sampling of the window blocks, the results are presented in Publication 6. Similar tests can be performed for the biased sampling scenario and are presented in Publication 7.

As the truncation error is negligible in the parameter range we use to study the erasure correction problem, we will solely rely on using the analytical results from this point on.

4.5.4 Form of the optimal degree distribution

Extensive numerical results using the Markovian analysis can be found in Publications 6 and 7. Only the highlights of those results are replicated here, most importantly focusing on the form of the optimal degree distribution for both uniform and biased sampling, and in addition the form of the

Table 4.3: Examples of optimal degrees for uniform sampling given by the analysis.

P	$p = 0.01$	$p = 0.05$	$p = 0.10$
0.1	15	13	10
0.2	15	12	10
0.3	13	11	9
0.4	12	10	9
0.5	11	10	9
0.6	11	9	9
0.7	10	9	9
0.8	10	9	8
0.9	10	9	8
1.0	9	8	8

Table 4.4: Analytical results using deterministic sampling patterns. Channel loss probability $p = 0.1$.

w	P	Q_r	pattern	d
12	0.1	0.0625	{1,1,1,0,1,0,1,1,0,1,1,1}	9
12	0.2	0.0392	{1,1,1,0,1,0,1,1,0,1,1,1}	9
12	0.3	0.0230	{1,1,1,0,1,0,1,1,0,1,1,1}	9
12	0.4	0.0119	{1,1,0,0,1,0,1,0,1,1,1,1}	8
12	0.5	0.0058	{1,1,0,0,1,0,1,0,1,1,1,1}	8
12	0.6	0.0025	{1,1,0,0,1,0,1,0,1,1,1,1}	8
12	0.7	$9.5 \cdot 10^{-4}$	{1,1,0,0,1,0,0,1,0,1,1,1}	7
12	0.8	$2.8 \cdot 10^{-4}$	{1,1,0,0,1,0,0,1,0,1,1,1}	7
12	0.9	$5.9 \cdot 10^{-5}$	{1,1,0,0,1,0,0,1,0,1,1,1}	7
12	1.0	$4.2 \cdot 10^{-6}$	{1,0,0,0,0,0,1,1,0,1,1,1}	6

sampling distribution for biased sampling.

The numerical studies suggest that the optimal form of the degree distribution for both cases is of a single-degree type. That is, although any degree distribution is allowed, the best performance in terms of lowest residual erasure probability Q_r possible is found to be attained using repair packets of a fixed degree. Example degrees when the stream blocks in the window are uniformly sampled are listed in Table 4.3.

Further, in addition to the fixed degree being optimal, with biased sampling the best possible performance is achieved using a deterministic sampling pattern. This means that $\boldsymbol{\omega} \in \{0, 1\}^w$. Examples of optimal degrees and patterns for biased sampling can be found in Table 4.4. The residual erasure probability Q_r obtained using the biased sampling scheme is found to be lower than that with uniform sampling. Therefore, the most effective method is to allow biased sampling of the source blocks. In Publication 7, the biased sampling method is also compared with Raptor coding as specified in [3GP07]. Our proposed biased sampling based coding turns out to give lower Q_r than the Raptor code does.

Note that these results on the optimality of the degree distribution and

sampling pattern are empirical in nature, although in the next section we try to reason why the single degree distributions give the best performance.

4.5.5 Discussion

An interesting finding regarding the form of the optimal sampling pattern is the fact that the pattern is dependent only on the used degree; for a given d the form of the pattern stays practically the same for any w and P. The optimal sampling pattern always includes ones at both ends of the window (i.e., at first and last elements). If this were not the case, then the window could be resized without loss of performance. Because enlarging the window size results in better performance in our window-based erasure correction method, it is always optimal to choose the first and last blocks inside a window. The middle positions are chosen in a more random way; there is no identifiable or intuitive pattern of ones in the middle positions.

The single-degree distribution works best for the sliding window scheme with continuous movement of one block at a time and probabilistic repair packet sending. As the decoder is assumed to take advantage only of such packets which immediately yield new decoded blocks, the decoding steps achieved using buffered packets are not taken into account in the analysis. In addition, only a couple of missing blocks in a window are assumed, a longer iteration chain would be worthless. The iteration is kept alive with degree-1 packets, and with low channel erasure probability the number of these packets is high compared to the window size. When an optimised fixed degree distribution is used, the probability that a sent repair packet will yield a new undecoded block after decoding is reasonably high (higher than with any other degree) and longer iterative decoding chains are unnecessary. When the channel conditions are static, and as the equilibrium is assumed on a very long stream, the same decision is made over and over again, and the optimal thing to do is to choose the single degree giving the highest probability of a new decoded block.

4.6 Summary

In this chapter we studied erasure correcting methods for data with real-time requirements, especially in a streaming environment. The methods studied were novel erasure correction strategies relying on a sliding window marking the section of source data relevant for the recipient, i.e., the non-expired data. All of the schemes studied were systematic, leaving the choice as to whether or not the erasure correction is taken advantage of to the recipient.

We started by considering dividing the window into two parts and moving the window in half-window step sizes. In this scheme, the repair packet degrees were chosen to form packets which, upon receipt, could immediately be used to decode a new stream block. The repair packets were sent deterministically after each window movement, and thus the window size used defines the code rate. The analysis was performed by making use of the independence of the number of missing blocks in each half of the window, and calculating steady-state results using fixed-point iteration.

The half-window strategy was compared to a continuous and probabilistic scheme in which the window is moved one step at a time and a repair packet is formed and sent with a probability P. Further, in this scheme, the repair packet is sampled by allowing different sampling probabilities p_i for every window position. The analysis was performed using a fixed-point iteration. However, the approximations made in the analysis always resulted in a fixed number of the newest blocks in a repair packet being chosen, making the window size somewhat irrelevant. The numerical evaluations indicated that the half-window scheme performs better.

As the degree-based sampling turned out to perform better, we continued by making a Markovian analysis of the continuous sliding window erasure correction with degree-based repair packet generation and probabilistic sending of the repair packets. The Markovian analysis was made possible by restricting the number of allowable packet erasures in a window to three; otherwise we would have faced similar state space explosion problems as in Chapter 3. This restriction, however, is a justified one for the scenarios we considered.

Both uniform and biased sampling were studied in the Markov model. Somewhat surprisingly, we found that a single-degree distribution gives the best possible performance. In addition to single-degree distribution, when biased sampling was used, a deterministic sampling pattern consisting of zeroes and ones turned out to give the best results.

5 SUMMARY

In this thesis we have presented a performance analysis and optimisation of different types of coding schemes based on the fountain principle. The codes that were analysed implement some or all of the different characteristics of a digital fountain. The digital fountain is an analogy to a conventional water-spraying fountain. In the fountain principle, the packets correspond to water drops which one or many recipients collect in their buckets. All of the packets are ideally statistically identical, like water drops, which will quench your thirst if you just collect enough of them. The fountain itself is the data-distributing server.

Scenarios such as multicasting and the dynamic joining and leaving of the receivers are easy to implement using codes realising the fountain principle. This makes these techniques interesting for many modern-day file transfer situations over the Internet. As the fountain codes work by encoding with a random algorithm, making the packets statistically equivalent and possibly redundant, the receiver needs to collect more than the absolute minimum number of packets in order to decode all the original data with a high probability. This is a key problem which needs to be addressed when employing fountain coding inspired schemes, as bad design leads to large overheads, resulting in inefficient channel and network use.

The first codes discussed in the context of a digital fountain were Tornado codes [LMSS01]. Before this, schemes like Reed-Solomon coding had been used to implement erasure correction, but not in quite similar context to implementing a digital fountain. However, these codes are not rateless and need to have their parameters, such as the number of generated encoding symbols, decided beforehand and matched to the channel loss statistics. The first universal fountain codes are the LT codes [Lub02], which are rateless and asymptotically optimal for any erasure channel. The encoding and decoding algorithms are simple and use only bitwise XOR operation. The current state-of-the-art fountain codes are the Raptor codes [Sho06], which are an extension to the LT codes, providing linear time encoding and decoding algorithm complexities. Despite the intriguing potential these codes have, they have not yet seen large-scale adoption.

Our own work in this thesis was divided into two parts. In the first part we discussed the performance analysis and optimisation of fountain codes for small file lengths. Most of the previously introduced analyses of fountain codes discuss only asymptotic properties, and have left room for the analysis of scenarios with small file lengths (the small number of blocks the file is divided into). First we optimised the degree distributions, which determine the performance, of LT codes for toy cases of three and four file blocks. We did an exact analysis and found the optimal degree distributions. As the analysis suffered from state space explosion, we introduced a combinatorial algorithm for larger cases of tens of file blocks and optimised the degree distributions. Even larger numbers of file blocks can be used with the simulation-based method, introduced after the exact analyses. Simulation samples with a specific degree distribution were used to construct

an estimate of the gradient direction where the distribution could be improved. This estimate was calculated using the concept of change of the probability measure, borrowed from importance sampling theory, allowing us to estimate the performance from simulation samples using a different degree distribution. The optimal form of the degree distributions of the LT codes found using these methods is the same: the first few degrees are important, and in addition some weight has to be given to some large degree. This is to guarantee that long chains of revealing novel blocks are kept alive during the decoding. Further, we analysed the sensitivity of the degree distributions and noticed that there are only a couple of principal directions of change which affect the performance in an observable way.

After the LT code optimisation, we turned to different settings for fountain principle-inspired codes. A random linear fountain (RLF) is a code, in which the packets are formed by sampling each of the k blocks with a probability $1/2$. As with the other methods, each sent packet can be interpreted to represent a linear equation. The receiver needs to collect k linearly independent equations and can then decode the original information by solving the linear system using Gaussian elimination. This strategy results in a very low overhead on average (1.6 packets). The problem is, however, that the decoding is done using a much slower algorithm than is typical or desirable for a fountain code. Therefore, to relieve the computational burden, we studied scenarios, in which RLF was divided into multiple parts. In order to get statistically identical encoded packets, first a part is picked at random and then RLF is used in that part to form a packet.

The problem with the division, however, is that when separate acknowledgements to indicate completed decoding in the different parts are not used, the sender keeps sending unnecessary packets to the already decoded parts, resulting in inefficiency. To counter this we proposed combining the RLF-generated packets into macropackets by using LT coding on top of the divided parts. In this way we still have statistically identical packets, but alleviate some of the inefficiency resulting from the division. The macropackets scheme was further compared to a data carousel-inspired way of sending packets. However, the latter method breaks the statistical identicality of the packets, as the packets are sent sequentially from each part. The data carousel scheme works better than the macropackets for independent loss rate below 0.2. The macropackets scheme outperforms it with higher loss rates and is a rateless method, true to the fountain principle.

If we forego the requirement for the coding to be time-independent, and have an estimate of the channel loss probability, how low an overhead can we get? We studied this question by introducing a code with systematic part followed by a sequence of repair packets. After all of the blocks have been sent once as such, the sender can calculate the distribution of the number of missing blocks on the receiver's side, the *belief* about the situation of the receiver. The repair packets are formed by using a greedy decision to choose the packet degree which would most probably advance the decoding at least one step further. The sender updates the belief about the state of the receiver after every sent packet on the basis of what the actual degree of the sent repair packet was. The code proposed for this setting achieves low overheads even for small file sizes. Although the performance

is good, the problem is that an estimate of the loss probability is needed. However, the scheme seems to be rather insensitive to small errors in the estimate.

The second part, describing our own work, considered settings in which we have a stream of data blocks. This kind of situation is typical for streaming audio and video applications with real-time requirements. We further used the assumption that a channel loss probability estimate is available. All of the settings were assumed to be ones in which a systematic code is preferred to a non-systematic one. A sliding window is used to mark the non-expired section of the stream. As the window moves, newly introduced blocks are sent as-is to guarantee the systematic operation. Two different viewpoints were taken: either the repair packets were generated by sampling using probabilities depending on the location of the block inside the window, or a repair packet degree was sampled first and then this number of blocks inside the window was picked at random.

In particular, first we considered the case where the window moves with the step size equalling a half-window and a degree is drawn separately for both of the halves. Then we allowed separate probabilities of each of the blocks being included in the repair packet. The half-window method, with degree-based sampling, turned out to be more efficient in terms of residual erasure probability, i.e., the probability that a block remains undecoded after it is dropped out of the sliding window. This encouraged us to study in more detail scenarios in which a degree is first chosen from a degree distribution and then the constituents are sampled inside the whole window. We presented a performance analysis of both uniform sampling inside the window and biased sampling. Of these, the biased sampling scenario gives a lower residual error probability. An interesting empirical observation was that the optimal degree distributions seem to be of a single-degree type. Furthermore, with biased sampling, the optimal repair packets are such that certain locations in the window are always sampled while others are not. That is, the optimal way seems to be the use of a deterministic sampling pattern.

We started the thesis by discussing 'real' fountain codes, but later on relaxed the strict requirement of fulfilling all of the ideal properties, e.g., by abandoning the time-independence requirements and assuming knowledge of the channel loss probability p. As an example, we studied schemes in which the data are initially sent as-is once, i.e., systematic coding, followed by repair packets. Thus, not all of the packets that are sent are statistically identical and the start of the initial round is a clear point in which the receivers should start receiving packets from the beginning. Nevertheless, typical repair packet construction relies on the same encoding algorithm as is used with LT codes and only bitwise XOR operation is used. An interesting observation is that a degree distribution similar to that used in plain LT coding is not required, but a better performance is achieved with strategies such as a sequence of packet degrees or even a deterministic sampling pattern.

The analyses and results in this thesis show that there are efficient ways to perform fountain-type coding in different settings with algorithms which are simple to understand and implement. It is the task of the implementer,

by taking into account the requirements and nature of the application being used, to decide which properties of the ideal fountain can be sacrificed and choose an appropriate fountain coding method.

This thesis has touched the surface of fountain coding analysis. There remains numerous research topics to be pursued. For example, the optimal degree distributions for general k for LT coding is an open question. Another interesting question is how well fountain-coding-inspired methods could co-operate with common transport layer protocols, such as TCP. The actual application potential of fountain codes was not discussed at all. Further, the optimisation methods presented in the thesis could be improved and refined to work for larger file sizes and more general situations.

6 AUTHOR'S CONTRIBUTION

Publication 1

The publication is a joint work between the authors. The ideas of modelling the LT process as a Markov chain and the combinatorial approach came from Prof. Jorma Virtamo and Dr. Esa Hyytiä. The present author has drafted the publication with the exception of Section IV.

Publication 2

The publication is a joint work between the authors. The present author has carried out all of the numerical evaluations, developed the optimisation algorithm and written the paper. The original idea of the importance sampling based estimator came from Dr. Esa Hyytiä.

Publication 3

The publication is a joint work between the authors.

Publication 4

The publication is a joint work between the authors. The idea for greedy optimisation came from the present author. The present author has done the numerical evaluations and drafted the paper.

Publication 5

The publication is a joint work between the authors. The idea for fixed point iteration came from Prof. Jorma Virtamo. The present author has performed the numerical evaluations and drafted the paper.

Publication 6

This publication is a joint work between the authors. The publication was drafted and all of the performance studies were conducted by the present author.

Publication 7

This publication is an independent work of the author.

REFERENCES

[3GP07] 3GPP. TS 26.346 v.7.6.0, technical specification group services and system aspects, multimedia broadcast/multicast service (mbms), protocols and codecs. Technical report, 3rd Generation Partnership Project, 3GPP, December 2007. available at www.3gpp.org, accessed 27.5.2009.

[AdP] E. Altman and F. de Pellegrin. Forward correction and fountain codes in delay tolerant networks. http://arxiv.org/abs/0808.3747 , accessed 7.6.2009.

[AKS08] Salah A. Aly, Zhenning Kong, and Emina Soljanin. Fountain codes based distributed storage algorithms for large-scale wireless sensor networks. In *Proceedings of the 7th international conference on information processing in sensor networks*, pages 171–182, April 2008.

[BC08] E. A. Bodine and M. K. Cheng. Characterization of luby transform codes with small message size for low-latency decoding. In *Proceedings of IEEE International Conference on Communications, ICC'08*, pages 1195–1199, May 2008.

[BCG+07] C.O. Mattia Bogino, Pasquale Cataldi, Marco Grangetto, Enrico Magli, and Gabriella Olmo. Sliding-window digital fountain codes for streaming of multimedia contents. In *Proceedings of IEEE International Symposium on Circuit and Systems'07, ISCAS'07*, May 2007.

[BCMR04] J.W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Transactions in Networking (TON)*, 12(5):767–780, October 2004.

[BDS07] Amos Beimel, Shlomi Dolev, and Noam Singer. RT oblivious erasure correcting. *IEEE/ACM Transactions on Networking*, 15(6):1321–1332, December 2007.

[BGT96] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding. *IEEE Transactions on Information Theory*, 42:1732–1736, November 1996.

[BKK+95] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. Technical report, International Computer Science Institute, 1995.

[BLM02] J.W. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1528–1540, 2002.

[BLMR98] J.W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review*, 28(4):56–67, 1998.

[BSS93] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons., Inc., 2nd edition, 1993.

[Che] J. Chesson. A non-central multivariate hypergeometric distribution arising from biased sampling with application to selective predation. *Journal of Applied Probability*, 13:795–797.

[Con01] J. Considine. Generating good degree distributions for sparse parity check codes using oracles. Technical Report BUCS-TR 2001-019, Boston University, 2001.

[CRZ08] P. Casari, M. Rossi, and M. Zorzi. Fountain codes and their application to broadcasting in underwater networks: performance and relevant tradeoffs. In *Proceedings of the 3rd ACM international workshop on wireless network testbeds, experimental evaluation and characterization*, pages 11–18, San Francisco, California, USA, September 2008.

[CT91] Thomas Cover and Joy Thomas. *Elements of Information Theory*, volume 41 of *Wiley Series in Telecommunications*. John Wiley & Sons, Inc., 1991.

[Did] Frederic Didier. Efficient erasure decoding of Reed-Solomon codes. http://arxiv.org/abs/0901.1886v1 , accessed 18.9.2009.

[DPR06] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Distributed fountain codes for networked storage. In *Proceedings of the 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2006*, pages 1149–1152, May 2006.

[DWR07] Alexandros G. Dimakis, Jiajun Wang, and Kannan Ramchandran. Unequal growth codes: Intermediate performance and unequal error protection for video streamin. In *Proceedings of the 9th IEEE Workshop on Multimedia Signal Processing, MMSP 2007*, pages 107–110, October 2007.

[Eli55] Peter Elias. Coding for two noisy channels. In *Proceedings of the 3rd London Symposium on Information Theory*, pages 61–76. Buttersworth's Scientific Publications, September 1955.

[Ell63] E.O. Elliott. Estimates of error rates for codes on burst-error channels. *Bell Systems Tech. Jouranl*, 42:1977–1997, September 1963.

[Fog08] Agner Fog. Calculation methods for wallenius' noncentral hypergeometric distribution. *Communications in Statistics - Simulation and Computation*, pages 258–273, 2008.

[Fro01] Pascal Frossard. FEC performance in multimedia streaming. *IEEE Communication Letters*, 5(3):122–124, March 2001.

[Gal62] Robert Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962.

[GBGC09] D. Gomez-Barquero, D. Gozalvez, and N. Cardona. Application layer FEC for mobile TV delivery in IP datacast over DVB-H systems. *IEEE Transactions on Broadcasting*, 55(2):396–406, June 2009.

[Gil60] E.N. Gilbert. Capacity of a burst-error channel. *Bell Systems Tech. Jouranl*, 39:1253–1266, September 1960.

[Hui96] Christian Huitema. The case for packet level FEC. In *Proceedings of the IFIP 5th International Workshop on Protocols for High-Speed Networks*, pages 109–120, Sophia Antipolis, France, October 1996.

[JS05] H. Jenkac and T. Stockhammer. Asynchronous media streaming over wireless broadcast channels. In *Proceedings of IEEE International Conference on Multimedia and Expo, ICME 2005*, pages 1318–1321, Amsterdam, The Netherlands, July 2005.

[KFMR05] Abhinav Kamra, Jon Feldman, Vishal Misra, and Dan Rubenstein. Data persistence in sensor networks: Towards optimal encoding for data recovery in partial network failures. *SIGMETRICS Performance Evaluation Review*, 33(2):24–26, September 2005.

[KHF06] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006.

[KLS04] R. Karp, M. Luby, and A. Shokrollahi. Finite length analysis of LT codes. In *Proceedings of International Symposium on Information Theory, ISIT 2004*, page 39, June 2004.

[KMFR06] Abhinav Kamra, Vishal Misra, Jon Feldman, and Dan Rubenstein. Growth codes: Maximizing sensor network data persistence. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM'06*, pages 255–266, Pisa,Italy, August 2006.

[Li05] J. Li. The efficient implementation of Reed-Solomon high rate erasure resilient codes. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'05*, volume 3, 2005.

[LMA98] M. Luby, M. Mitzenmacher, and Shokrollahi A. Analysis of random processes via and-or tree evaluation. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 364–373, 1998.

[LMS+97] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. pages 150–159, 1997.

[LMSS98] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. A. Spielman. Analysis of low density codes and improved designs using irregular graphs. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 249–258, 1998.

[LMSS01] Michael Luby, Michael Mitzenmacher, Amin Shokrollahi, and Daniel Spielmann. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, February 2001.

[LSW08] M. Luby, T. Stockhammer, and M. Watson. IPTV systems, standards and architectures: Part II - application layer FEC in IPTV services. *IEEE Communications Magazine*, 46(5):94–101, May 2008.

[LSWS07] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery. RFC 5053 (Proposed Standard), October 2007.

[Lub02] Michael Luby. LT codes. In *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002*, pages 271–280, 2002.

[Mac03] David J.C. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[Mac05] David J.C. MacKay. Fountain codes. *IEE Proceedings Communications*, 152(6):1062–1068, December 2005.

[Max75a] N.F. Maxemchuk. Dispersity routing. In *Proceedings of ICC'75*, pages 41–10,41–13, San Francisco CA, USA, 1975.

[Max75b] N.F. Maxemchuk. *Dispersity routing in store-and-forward networks*. PhD thesis, University of Pennsylvania-Electronic Dissertations, 1975.

[May02] Peter Maymounkov. Online codes. Technical report, New York University, October 2002.

[McA90] A. J. McAuley. Reliable broadband communication using a burst erasure correcting code. In *Proceedings of the ACM symposium on communications architectures and protocols, SIGCOMM'90*, pages 297–306, Philadelphia, Pennsylvania, USA, 1990.

[MM03] Peter Maymounkov and David Mazierès. Rateless codes and big downloads. In *In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems, IPTPS'03*, Berkeley, California, USA, 2003.

[MN97] David J. C. MacKay and R.M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics letters*, 33(6):457–458, 1997.

[MS77] F.J. Macwilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*, volume 16. North Holland Mathematical Library, 1977.

[NB96] Jörg Nonnenmacher and Ernst Biersack. Reliable multicast: where to use FEC. In *Proceedings of the IFIP 5th International Workshop on Protocols for High-Speed Networks*, pages 134–148, Sophia Antipolis, France, October 1996.

[NYH07] T. D. Nguyen, L. L. Yang, and L. Hanzo. Systematic luby transform codes and their soft decoding. pages 67–72, October 2007.

[PB05] James S. Plank and Adam L. Buchsbaum. Small parity-check codes – exploration and observations. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 326–335, June 2005.

[Pla05] James S. Plank. Assessing the performance of erasure codes in the wide-area. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks, DSN'05*, pages 182–187, Washington, DC, USA, 2005. IEEE Computer Society.

[PLL+04] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. FLUTE – file delivery over unidirectional transport. RFC 3926 (Experimental), October 2004.

[Pos80] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[PT03] J. S. Plank and M. G. Thomason. On the practical use of ldpc erasure codes for distributed storage applications. Technical Report CS-03-510, University of Tennessee, September 2003.

[Rab89] M.O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348, 1989.

[Riz97a] L. Rizzo. On the feasibility of software FEC. 1997. available at: http://www.iet.unipi.it/ luigi/softfec.ps , accessed 5.5.2009.

[Riz97b] Luigi Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM SIGCOMM Computer Communication Review*, 27(2):24–36, 1997.

[Ros00] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, 7th edition, 2000.

[RS60] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, pages 300–304, 1960.

[RU01] T. J. Richardson and Rüdiger Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2), February 2001.

[RU08] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory*. Cambdrigde University Press, 2008.

[Rub97] Reuven Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal on Operational Research*, 99(1):89–112, May 1997.

[Sha48] Claude Shannon. A mathematical theory of communication. *Bell Systems Tech. J.*, 27:379–423, 1948.

[Sho06] Amin Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, June 2006.

[Sii08] Sebastian Siikavirta. Toistokorjausalgoritmit multimedian siirrossa matkapuhelinverkoissa. Master's thesis, University of Helsinki, 2008.

[SM90] N. Shacham and P. McKenney. Packet recovery in high-speed networks using coding and buffer management. In *Proceedings of IEEE Infocom'90*, pages 124–131, June 1990.

[Tir06] Tuomas Tirronen. Optimizing the degree distribution of LT codes. Master's thesis, Helsinki University of Technology, 2006.

[Wal63] K.T. Wallenius. *Biased Sampling: The Non-central Hypergeometric Probability Distribution*. PhD thesis, Stanford University, 1963.

[WCF06] J.-P. Wagner, J. Chakareski, and P. Frossard. Streaming of scalable video from multiple servers using rateless codes. In *Proceedings of IEEE International Conference of Multimedia and Expo, ICME 2006*, pages 1501–1504, Toronto, Canada, July 2006.

[WHZ+01] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M. Peha. Streaming video over the internet: Approaches and directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3), March 2001.

[Wol] Wolfram. Mathematica. http://www.wolfram.com , accessed 27.5.2009.

[YCLX08] Wending Yao, Lijia Chen, Hui Li, and Hongguang Xu. Research on fountain codes in deep space communications. In *Proceedings of Congress on Image and Signal Processing, CISP'08*, pages 219–224, May 2008.

[YP08] Xiaojun Yuan and Li Ping. On systematic LT codes. *Communications Letters, IEEE*, 12(9):681–683, September 2008.